Česká společnost uživatelů otevřených systémů EurOpen.CZ
Czech Open System Users' Group
www.europen.cz

41. konference

Sborník příspěvků

Hotel Mlýn
Vílanec u Jihlavy
14.–17. října 2012

Programový výbor:

Vladimír Rudolf, ZČU Plzeň
Jan Kynčl, Dempsey & Clark, s. r. o.
Jiří Sitera, ZČU Plzeň

Sborník příspěvků z 41. konference EurOpen.CZ, 14.–17. října 2012

Plzeň 2012. První vydání.
Editor: Vladimír Rudolf
Sazba a grafická úprava: Ing. Miloš Brejcha – Vydavatelský servis, Plzeň
e-mail: servis@vydavatelskyservis.cz

Tisk: Typos, tiskařské závody, s. r. o.
Podnikatelská 1160/14, Plzeň

Příspěvky neprošly redakční ani jazykovou úpravou.

ISBN 978-80-86583-24-2

Obsah

Pavel Křižanovský
IT infrastruktura jako základ inteligentní budovy 5

Tomáš Novotný, Aleš Marvan, Josef Hájek, Martin Drahanský
Vestavěný Linux pro inteligentní dům 15

Radek Semrád
SmartGrid & Smart metering 31

Václav Novák
Managing chytrých měřicích systémů 37

Soňa Netoličková
Smart metering – nová koncepce měření! 45

Eduard Janeček, Petr Janeček
Modely a metody výpočtů sítí NN zahrnující fotovoltaické zdroje 49

Tomáš Vondra
PostgreSQL vs. SSD 61

Pavel Stěhule
Sloupcové databáze 79

Petr Jiroušek
Proti proudu času aneb Oracle flashback technologie? 85

Jozef Mlích, Aleš Láník, Pavel Zemčík
Vývoj aplikací v Qt pro mobilní zařízení 93

Štěpán Bechynský
Windows Phone 8 a Windows 8 – sdílení kódu 105

Matěj Konečný
Android 107

IT infrastruktura jako základ inteligentní budovy

Pavel Křižanovský

E-mail: pkrizano@cisco.com

1 Co jsou inteligentní budovy?

Pojmem Inteligentní budova bývá nejčastěji označována budova, která kombinuje technické i organizační prostředky za účelem maximálního uspokojení potřeb uživatelů, provozovatelů i vlastníků budovy. Tyto cíle mohou zahrnovat mimo jiné např. komfort prostředí (teplota, osvětlení, poskytované služby apod.), zvýšení produktivity činností v budově prováděných (např. zaměstnanci v kancelářských prostorách) a samozřejmě i efektivitu provozu budovy a rychlou návratnost vložených investic.

Z technologického hlediska jsou pod pojmem inteligentní budova často chápány budovy s nějakou formou centralizovaného řízení. Současné komerční budovy jsou vybaveny velkým množstvím technologických systémů: vytápění, ventilace, chlazení, osvětlení, výtahy, kamerové systémy, EPSka, systémy řízení fyzického přístupu apod. Donedávna byly tyto systémy budovány nezávisle na sobě. V posledních letech dochází k vzájemnému propojování, integraci a konsolidaci těchto systémů a jejich jednotnému řízení. Moderní IT prostředky včetně komunikačních sítí hrají v tomto procesu stále větší roli.

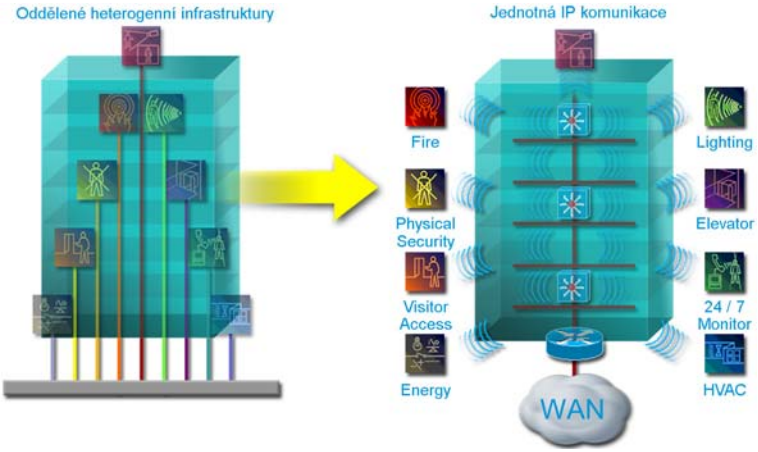
2 Historie a trendy

Historie inteligentních budov spadá zhruba do 70. let minulého století, technologické kořeny pocházejí ze systémů řízení, automatizace a optimalizace výrobních procesů. První větší rozvoj řízení systémů pak nastal hlavně v průběhu 80. let. Základem řešení se stalo využití PLC kontrolerů (Programmable Logic Controllers), které byly později nahrazeny distribuovanými mikroprocesory. Řídící systémy se začínají souhrnně označovat jako BAS (Building Automation Systems), EMS (Energy Management Systems), později BMS (Building Managements Systems).

Zpočátku byly jednotlivé technologie řízení velmi heterogenní. Každý výrobce typicky používal propriteární systém, proprietární propojovací mechanismy jednotlivých součástí daného systému a proprietární komunikační protokoly určené pouze pro danou funkcionalitu (řízení klimatizace, světel apod.). Jednotlivé technologie v budově byly budovány jako naprosto oddělené BAS systémy s vlastními řídícími pracovišti. Propojení systémů v takovém případě probíhalo (a někdy i stále probíhá) až na úrovni lidské obsluhy.

V posledních desetiletích se začaly využívat integrované BMS systémy, které umožňují zastřešit tato často stále ještě heterogenní prostředí pod jednu správu. BMS systémy jsou dnes budovány na bázi IT informačních systémů běžících v prostředí standardního IT prostředí na TCP/IP síti, s heterogeenními subsystémy pro jednotlivé technologie budov komunikují typicky pomocí bran (gateway), umožňujících překlady protokolů a fyzických kanálů (sběrnic) na společnou platformu (např.nějaký komunikační protkol nad IP).

Pro lepší interoperabilitu systémů začaly cca v 90. letech dvacátého století vznikat standardy (např. LON – Local Operating Networks, EIB – European Installation Bus a jeho následník KNX, BACnet – Building Automation and Control Network, ad.), nicméně standardů je stále relativně velké množství, jsou podporovány různými uskupeními výrobců, takže o nějaké rychlé konvergenci k jedné skupině univerzálních end-to-end standardů pro plnou interoperabilitu však zatím bohužel mluvit nelze.



Obr. 1

Za viditelný a pochopitelný trend lze však označit integraci roztříštěných systémů do jednoho transportního komunikačnho prostředí na bázi LAN/WAN TCP/IP sítí a jejich těsné propojení s tradičními informačními systémy (facility

management, logistika, personalistika apod.) a dalšími existujícími komunikačními prostředky (hlas, telepresence/videokonference, digital signage, ...).

Dalším velkým viditelným trendem, který je viditelný především v posledních letech, je snaha o snižování energetických nákladů na provoz budovy. Hlavním hybatelem jsou samozrejmě vlastní provozní náklady za energie, nezapomínejme však i na velkou úlohu marketingu. Budovy, které jsou označovány jako šetrné, nebo „green", jsou dnes často pro nájemce přitažlivější, takřka veškerá nová výstavba významnějších komerčních budov se provádídle některé šetrné certifikace (LEED, BREEM, DGNB, SB Tools ad.). Obzvláště nadnárodní korporace, které staví nebo pronajímají pobočky v ČR, dnes certifikace berou jako běžnou a nepostradatelnou věc v rámci svého zeleného marketingu. Nízká energetická náročnost a šetrnost k životnímu prostředí se tak promítá takřka do všech technologických systémů budov.

3 Role IT infrastruktury

Informační technologie hrají v moderních komerčních budovách v dnešní době dvě základní role:

- IT jako business nástroj
- IT jako sjednocující platforma pro ostatní technologie v budovách

Při pohledu na IT jako business nástroj máme na mysli především tradiční IT infrastrukturu, ať již informační systémy a aplikace podporující základní fungování uživatele/nájemce v budově, ale i další nástroje nezbytné pro chod firmy či organizace, jako je vzájemná datová hlasová a video komunikace uživatelů s veškerou síťovou infrastrukturou, která je k tomu potřeba. Z tohoto hlediska je v rámci budovy zapotřebí vidět v IT velkého konzumenta energie.

Pohled na IT jako na sjednocující platformu pro ostatní technologické celky v budovách jsme již částečně zmínili. Jednak jsou dnes tyto technologie (měření a regulace pro ovádání vytápění, chlazení, ventilace, osvětlení, ...) propojovány přes Building Management Systémy navzájem a dále do tradičních informačních systémů pro facility management apod., ale pro účel tohoto článku je daleko zajímavější fakt, že je pro komunikaci stále více využívána LAN/WAN infrastruktura, slangově označovaná jako „technologická LANka". Dnes jsou typicky technologické LAN sítě stavěny jako sítě dedikované pouze pro účely systémů budov, je však takovýto design optimální z hlediska nově požadovaných funkcionalit a provozních nákladů ?

Oběma těmto pohledům se budeme věnovat v následujících odstavcích.

3.1 IT jako konzument energie v budovách

Důležitost IT technologií prudce narůstá a logicky narůstá i podíl na celkové spotřebě energie. Dle různých studií dnes IT spotřebová cca čtvrtinu celkové energie v administrativních budovách a řadí se tak na druhé místo za systémy vytápění, větrání a klimatizace.

Obr. 2

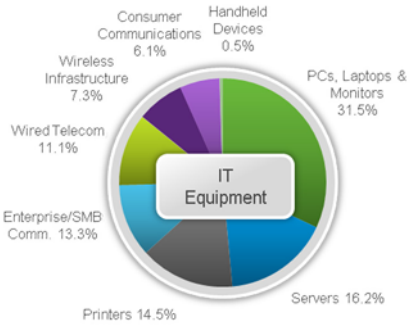
A jakou roli v rámci energetické spotřeby celého IT hraje samotná LAN infrastruktura v podobě aktivních prvků? Samotná LAN infrastruktura je samozřejmě spotřebitelem energie, nicméně může sloužit i pro napájení dalších IT komunikačních prvků typu bezdrátové přístupové body (AP), IP kamery, IP telefony apod. V této oblasti dochází k rozvoji technologií a standardů, před deseti lety bylo dle standardu IEEE 802.3af Power over Ethernet možné napájet z LAN prvků zařízení o příkonu cca 15 W na LAN port, dnes již existují technologie s možností napájet zařízení až 60W na port (např. technologie Cisco Universal POE). Spotřebu takovýchto zařízení lze tak z aktivních prvků LAN velmi jidnoduše měřit a řídit.

Nicméně to rozhodně není vše. Jak je zřejmé z následujícího obrázku, takto napájená a řízená zařízení tvoří společně s vlastními aktivními prvky cca třetinu spotřeby energie z celého „koláče spotřeby IT“. Největším konzumentem v této oblasti jsou ve skutečnosti počítače – osobní počítače jednotlivých uživatelů a datové servery – konzumují necelou polovinu.

Je tedy zřejmé, že pokud chceme dosáhnout výrazných úspor energie spotřebované v IT, je třeba pracovat s technologiemi zaměřujícími se na celé řešení IT včetně počítačů, tiskáren, telefonů apod.

Je tedy pochopitelné, že se takovýmito globálními koncepty pro ušetření energií v IT zabývají velcí výrobci a dodavatelé IT infrastruktury.

Příkladem může být architektura EnergyWise, kterou před lety vyvinula společnost Cisco Systems. Jedná se o technologii, která prostřednictvím pře-



Zdroj: Gartner Dataquest, Forecast of IT Hardware Energy Consumption, Worldwide, 2005-2012

Obr. 3

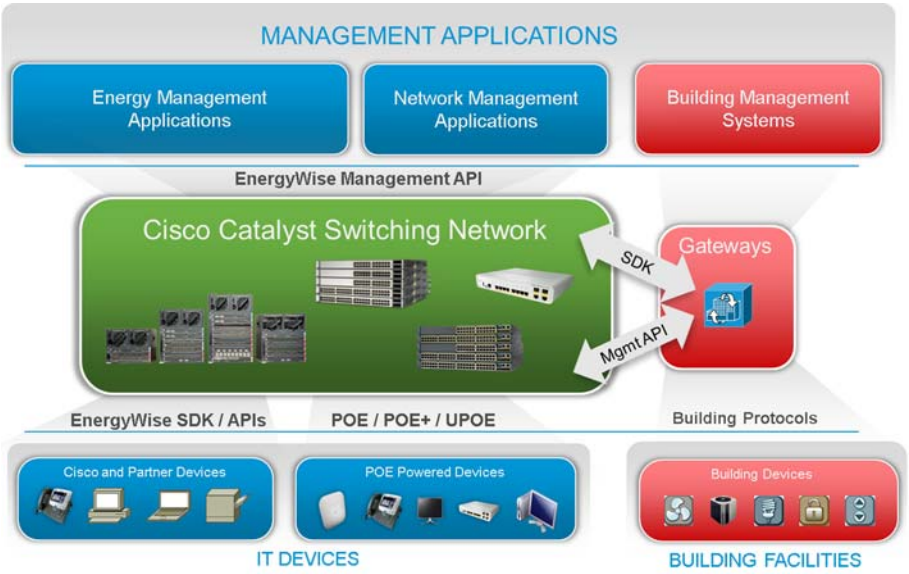
pínačů a směrovačů měří, řídí a optimalizuje spotřebu energie v celé IT infrastruktuře, a to nejen v rámci jedné konkrétní budovy, ale potenciálně v celé firmě/organizaci, tedy v budovách, které mohou být rozmstěny po celé zeměkouli. Měřením zjistíte současnou spotřebu energie a na základě analýzy spotřeby (obvykle prostřednictvím specializované aplikace) definujete pravidla, na jejich základě je pak řízena spotřeba připojených zařízení. Tato pravidla přenese a vynutí síťová infrastruktura. Měřením v síti s definovanými pravidly pak posoudíte dosažené úspory. Takto lze například automaticky uspávat, vypínat či zapínat uživatelská PC, která tak nemusí zbytečně běžet přes noc, kdy na nich typicky nikdo nepracuje.

Obr. 4

Řízení spotřeby pomocí technologií typu EnergyWise se neomezuje pouze na klasická IT zařízení. Přes definovaná rozhraní, s pomocí speciálních aplikací

(bran), je možné měřit a řídit spotřebu vytápění, ventilace, klimatizace a dalších systémů. Zkušenost ukazuje, že je možné dostat přes LAN síť pod kontrolu až 70 procent spotřeby energie v budově. Úspory energie a nákladů pak mohou být velmi výrazné (desítky procent).

Příklad globální architektury Cisco EnergyWise je zachycen na obrázku 5.

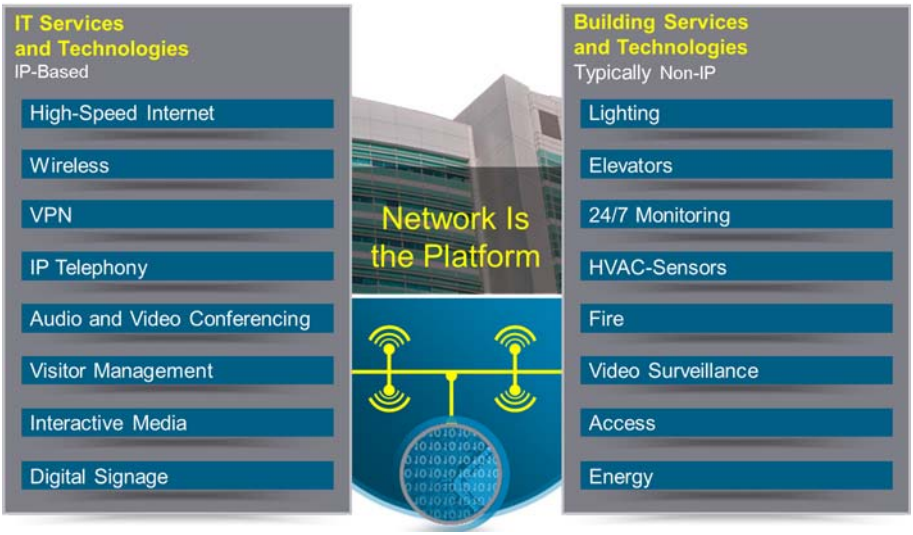


Obr. 5

Obdobné architektury již dnes nejsou doménou osamocených firem snažících se hlava nehlava prosadit nějaký uzavřený koncept. Jde o snahu vytvořit obecnou, otevřenou architekturu umožňující kromě energetických úspor v IT propojit přes IT infrastrukturu i ostatní systémy budov. Například technologii EnergyWise podporuje dnes již cca 100 firem od výrobců počítačů, chytrých elektrických zásuvek, LED osvětlení až po dodavatele specializovaných bran umožnujících přenos energetických informací mezi IT a tradičními BMS systémy.

3.2 IT infrastruktura jako sjednocující platforma pro systémy v budovách

Jak již bylo řečeno v předchozích kapitolách, tradiční systémy budov stále častěji využívají pro komunikaci v rámci daného systému i mezi systémy navzájem „datové“ LAN sítě na bázi technologie Ethernet a TCP/IP, tedy těch samých komunikačních technologií, které jsou masivně využívány pro tradiční IT systémy. Mluvíme zde tedy o konvergenci a integraci obou těchto infrastruktur.



Obr. 6

Technologické LAN sítě budov historicky byly a většinou stále ještě v některých případech jsou stavěny jako dedikované datové sítě na vyhrazené LAN infrastruktuře (separátní přepínače apod.). Velmi často se jedná pouze o základní přepínanou infrastrukturu v primitivní L2 topologii. Ukazuje se však, že takovýto přístup k výstavbě technologických sítí již dnes nesplňuje stále rostoucí požadavky, mezi které patří:

- Vysoká dostupnost – centralizace řízení technologií pomocí jednotného Building Management Systému by v případě výpadku části infrastruktury mohla například vést k prudkému zhoršení podmínek v budově (např. výpadek klimatizace) a ke zvýšeným provozním nákladům (nefunkční regulační mechanismy – světla, vytápění apod.) či k bezpečnostním rizikům (nefunkční kamerový systém apod.)

- Škálovatelnost – infrastruktura musí být připravena na nárůst počtu připojovaných technologických zařízení (nový nájemce budovy vyžaduje změnu v řídících systémech, pravidelná modernizace, apod.)

- Vysoká úroveň zabezpečení – ochrana proti neoprávněným přístupům k řídícím systémům budovy, ochrana před DoS útoky nového typu, např. snaha o „vymrazení“ nájemců budovy), ochrana dat jednotlivých technologických systémů (např. kamerové záznamy, přístupové informace zaměstnanců apod.)

- Řádově vyšší přenosové kapacity – masivní nárůst kamerových systémů, HD kvalita kamerových toků, reklamní multimediální panely (Digital Signage) apod.

- Mobilita – nástup bezdrátových senzorů (Smart Metering) vyžadujících bezdrátovou infrastrukturu

- Power over Ethernet – nové senzory již mohou být napájeny přímo z infrastruktury pomocí stardů (např. IEEE 802.3af či 802.3at)

- Bezpečný přístup na Internet a z Internetu – jedním z trendů v rámci Facility Managementu je vzdálená správa, vyžadující vzdálený přístup do technologické sítě z Internetu (outsourcing správy budovy externí firmou „na dálku“), cloudové řešení energetické správy (Remote Energy Management as a Service).

- Nutnost propojení na IT systémy – napojení do běžných informačních systémů (účetnictví, personalistika, výrobní procesy, ...) či využití nových sofistikovaných energy management funkcí infrastruktury popsané v minulém odstavci.

Ačkoli z hlediska technologických systémů budov mohou tyto požadavky vypadat jako nové, čtenář snadno nahlédne, že v tradičním IT prostředí jsou tyto požadavky s úspěchem řešeny naprosto běžně již poměrně velmi dlouhou dobu (∼desetiletí).

Jednou z možností, jak tyto požadavky pokrýt v rámci inteligentních budov, je tedy použít při výstavbě dedikovanou infrastrukturu vystavěnou na stejných principech jako v tradičním IT prostředí. Toto samozřejmě možné je, nicméně je to poměrně finančně náročné, neboť v podstatě buduji paralelní LAN infrastrukturu s rozšířenými funkcemi (bezpečnost, dostupnost, mobilita, škálovatelnost, napájení apod., napojení do WAN), přižemž stejnou úlohu mám již typicky vyřešenu v oblasti tradiční IT infrastruktury, bez které se v komerční budově neobejdu.

Logicky se zde tedy nabízí i druhá možnost, tedy využít již existující IT infrastrukturu, která typicky disponuje všemi výše zmíněnými funkcemi, a připojit do ní všechny technologické systémy budov. I tato možnost je realizovatelná, navíc na první pohled přináší řadu výhod spojených s řádově nižšími inestičními i provozními náklady do konvergované infrastruktury. Nemusí však být využitelná v každé situaci, je třeba minimálně vyřešit několik s tím spojených otázek.

- Logické oddělení sítí – k řídícím systémům budovy by neměli mít přístup běžní uživatelé IT. Toto je relativně snadno řešitelné různými virtualizačními, VPN a dalšími bezpečnostními technikami (VLANy, VRF – virtual routing a forvarding instance, MPLS VPN, přístupové filtry apod.).

- Oddělená správa IT a buidling prostředků – řešitelné obdobným způsobem, jako výše logickým oddělením infrastruktur. Nutno politicky vyřešit fakt, že IT oddělení poskytuje transportní služby pro technologické systémy, které operuje správce budovy.

- Dedikované pásmo pro technologické systémy budov – např. garance pásma pro kamerové systémy či pro řízení v reálném čase. I toto lze elegantně řešit pomocí standardních QoS mechanismů, kterými IT infrastruktura běžně disponuje.

- Politické hledisko – Nájemce prostor v budově může být subjekt naprosto nezávislý na firmě, která vlastní/spravuje danou budovu, a typicky si tak do budovy přináší i své vlastní IT prostředky na budově nezávislé. Obzvláště u větších organizací či i u menších poboček nadnárodních korporací je myšlenka sdílení či plné propojení infrastruktur budov a IT politicky zcela nemyslitelná, neproveditelná a často explicitně zakázaná bezpečnostními a organizačními směrnicemi dané organizace. Toto bývá největší oříšek a pro plnou konvergenci infrastruktur často tzv. show-stopper.

Takže ačkoli by plná konvergence a integrace obou typů infrastruktur technicky dávala smysl a přinesla by kromě elegantního vyřešení většiny výše uvedených požadavků i kýžené finanční efekty, je zřejmé, že ji zatím často z politických důvodů nelze realizovat. Alespoň částečně je však možné některé výše uvedené požadavky splnit pomocí hybridního modelu, kdy jsou sice infrastruktury budovy a IT budovány a provozovány odděleně, nicméně jsou spolu řízeným způsobem v jasně definovaných bodech bezpečně propojeny. Takto lze např. řešit společný přístup do Internetu, propojení BMS na IT informační systémy, bezdrátové sítě pro building systémy realizované na dedikovaném SSID v rámci wifi spravovaného IT apod.

4 Závěr

Svět technologických systémů v takzvaných inteligentních budovách prožívá bouřlivý vývoj, který je srovnatelný s rozvojem IT technologií. Dochází k postupné konsolidaci a standardizaci systémů, sjednocování správy a řízení a k postupnému přechodu na jednotnou transportní infrastrukturu využívající technologií na bázi Ethernet a TCP/IP.

Na IT infrastrukturu v komerčních/administrativních budovách se lze dívat z různých úhlů pohledu.

IT je poměrně velký konzument energie (typicky 25 % spotřeby budovy). Odpovědí na tento fakt je vznik technologií pro kompexní energetický management

celého IT prostředí s využitím prostředků chytré síťové infrastruktury s možností napojení do globálního systému řízení celé budovy.

Velkým trendem je též integrace a konvergence IT infrastruktury s infrastrukturou pro technologické systémy budov za účelem vyřešení nově kladených požadavků na building systémy (vysoká dostupnost, bezpečnost, propojení do Internetu, mobilita apod.) Plná integrace však není v dnešních podmínkách vždy proveditelná, hlavní pžekážkou bývají především politické aspekty.

Na závěr je však zapotřebí si uvědomit, že nové a nové technologie a sofistikované automatizované systémy mohou přinášet lepší podmínky pro uživatele budovy, výhody pro správce a vlastníka dané budovy, nicméně nejsou samospasitelné. Nepřítelem k úspěšnému nasazení se často může stát nikoli technologie sama, ale její nesprávné používání, složitost obslužných systémů či nedostatek znalostí jednotlivých uživatelů. Mějme tedy při realizaci inteligetních budov na paměti, že je nestavíme kvůli technologiím samotným, ale především kvůli lidem, kteří ji budou užívat ať již jako nájemci, či jako pronajímatelé a vlastníci.

Vestavěný Linux pro inteligentní dům

**Tomáš Novotný, Aleš Marvan,
Josef Hájek, Martin Drahanský**

E-mail: {inovottom,imarvan,ihajek,drahan}@fit.vutbr.cz

Abstrakt

S rozvojem výpočetní techniky, sítí a snižováním ceny integrovaných obvodů je dnes možné vytvořit velkou a levnou síť vestavěných počítačů. Tyto možnosti společně s požadavky na komfort v moderních obydlích vedou k rozvoji inteligentních domů. Příspěvek se nejprve zabývá popisem inteligentního domu a toho, jaké jsou vůbec důvody „inteligenci" doplňovat. V dalších částech jsou postupně probrány možnosti a využití Linuxu ve vestavěných systémech, protože implementace řídicí jednotky inteligentního domu je vestavěným systémem. Poslední část je věnována propojení těchto dvou témat. Jde o popis návrhu a tvorby centrální jednotky, která je postavena na Linuxu.

1 Inteligentní domy

Úvodní část se věnuje popisu inteligentního domu, jeho součástem, řízení a komunikaci mezi jednotkami.

1.1 Inteligentní dům

Moderní inteligentní domy jsou v současné době velmi populární a jejich uživatelé si je pořizují nejen pro zvýšení komfortu vlastního bydlení. Infrastruktura elektroniky řízených a akčních prvků rozmístěných po budově umožňuje monitorovat a ovládat takřka cokoliv, co se v něm a jeho bezprostřední blízkosti děje. Dnešní systémy domácí automatizace však nejsou již jednostranně zaměřené, ale lze si vybírat i podle zaměření, například:

- Úspora energie – takto zaměřený systém se specializuje na ušetření všech druhů energií a využívá se především v tzv. nízkoenergetických domech, kde je potřeba toky energií usměrňovat.

- Bezpečí – dům může hlídat sám sebe. Struktura vhodně rozmístěných bezpečnostních čidel detekuje podezřelé aktivity přímo v něm či jeho okolí. Ať už se jedná o požáry, zatopení sklepních prostor či úniku plynu. Tyto systémy také dokáží zastoupit EZS (elektronický zabezpečovací systém) a mohou přímo komunikovat s policií skrze pulty centrální ochrany.

- Komfort – bezesporu jeden z nejrozšířenějších důvodů, proč si uživatelé do svého domu „inteligenci“ pořizují. Umožňuje pak např. automaticky stahovat žaluzie na základě osvětlení, automaticky otevírat garážová vrata při příjezdu auta, regulovat teplotu v místnosti. Patří zde však i jednoduché úkony jako třeba automatické spuštění zavlažování nebo rozsvícení světla vypínačem. Použití těchto prvků v domě se běžně označuje termínem systémová integrace. Díky automatizaci je užívání takového domu výrazně jednodušší. Jedním tlačítkem lze při odchodu zhasnout světla, vypnout televizi, zatáhnout žaluzie, regulovat vytápění, aktivovat EZS a odpojit rizikovější spotřebiče jako sporák, rychlovarná konvice apod.

1.2 Struktura řízení inteligentního domu

Modelů pro řízení inteligentní budovy existuje více. Vzhledem k povaze této práce, kdy je využíván Linux v centrální jednotce, se však zaměříme na scénář řídicí jednotky (Master) a řízených podjednotek (Slave).

1.2.1 Centrální jednotka

Řídicí jednotka je jakýmsi mozkem celého systému. Je to arbitr, který komunikuje s ostatními jednotkami. Musí být tedy propojena se všemi ostatními moduly v systému. Uživatel většinou pomocí nějakého programu, aplikace na PC nebo chytrého telefonu definuje program, podle kterého se bude řízení domu chovat. Informace ze senzorů jsou pak zasílány do centrální jednotky, která je vyhodnotí a provede příslušné výstupní akce. Definuje tak vztahy mezi vstupními a výstupními jednotkami. Možnosti takového systému jsou takřka neomezené a záleží jen na fantazii a potřebě uživatele.

Dále je potřeba definovat terminologii vstupních a výstupních jednotek, užívanou v souvislosti s inteligentními domy.

- Aktor – výstup, který provádí daný úkol na základě příkazu z centrální jednotky.

- Senzor – vstup, na jehož základě řídicí jednotka může definovat nějakou akci.

Řídicí jednotka je v podstatě programovatelný logický automat, jehož programovací rozhraní je uživateli prezentováno ve formě akce – reakce. Toto rozhraní

je často definováno jako aplikace, ve které si uživatel definuje skelet svého domu včetně jednotlivých místností, do kterých pak umístí senzory a aktory. Tím se stává programování centrální jednotky velice jednoduché a pohodlné.

1.2.2 Ostatní jednotky

V zásadě se dají podružné jednotky rozdělit na senzory a aktory podle toho, zda jsou určeny pro monitorování nebo provedení nějaké akce. Pro představu jsou zde uvedeny příklady několika jednotek a k čemu slouží:

- Spínač (senzor) – jeden z nejzákladnějších vstupů. Ve spojení s centrální jednotkou jej lze použít ke spínání takřka čehokoliv.
- Relé (aktor) – jeden z nejzákladnějších výstupů. Umožňuje zapnout/vypnout připojený spotřebič.
- Hladinový snímač (senzor) – monitoruje výšku hladiny kapaliny. Využití v bazénech, sklepech (při zatopení) apod.
- Termostat (senzor) – slouží k nastavení teploty. Pozor, nejedná se však o aktor. O samotnou regulaci se stará řídicí jednotka.
- Spínací hodiny (aktor) – zapne/vypne připojený spotřebič v nastaveném čase. Je možné je také realizovat ve spojení relé a řídicí jednotky, která jej spíná ve zvolených intervalech.

1.3 Komunikace mezi jednotkami

Senzory a aktory musí být nějakým způsobem propojeny s řídicí jednotkou pomocí vhodné sběrnice. Napájení je však kromě výjimek přivedeno samostatnými vodiči. Dále jsou uvedeny nejpoužívanější standardy pro komunikaci v inteligentním domě.

LonWorks LON, Local Operating Network [2] – sběrnice vyvinutá v letech 1989–1992 firmou Echelon Corporation ve spolupráci s firmami Toshiba a Siemens. Jako fyzická vrstva může sloužit prakticky cokoliv – kroucená dvoulinka, síťové rozvody, RF, optika, RS-485 atd. Síť může být rozdělena na domény. Využívá p2p architektury (komunikace uzel–uzel). Základem je tzv. inteligentní node. Ten je založen na speciálních mikroprocesorech Neuron chip, na nichž běží LongTalk protokol, který řídí zprávy a směruje samotné zpráv. Tento protokol tvoří firmware každého Neuron chipu. V jeho EEPROM je také uložena 48 bitová adresa jako jedinečný identifikátor.

BACnet Building Automation and Controls Network [3] – je od roku 2003 evropským standardem v rámci CEN s označením ISO 16 484-5. Skládá se ze specifické struktury BACnet objektů, služeb, přenosového protokolu (síťová vrstva) a fyzická vrstva. Jako fyzická vrstva se běžně používá RS-485 nebo Ethernet. Je založena na objektové reprezentaci a organizaci fyzických vstupů a výstupů. Objekt může reprezentovat samostatný fyzický uzel, nebo i skupinu uzlů tvořících pohromadě nějakou funkci. Každý BACnet objekt je definován sadou vlastností, které popisují jeho chování a řídí provoz.

KNX Konnex bus [4] – sdružuje tři existující sběrnice EIB (European Installation Bus), BatiBus a EHS (European Home System). Umožňuje tak komunikaci mezi přístroji více výrobců. Umožňuje komunikovat rychlostí až 32 kbit/s v závislosti na použitém přenosovém médiu a datové pakety mohou mít velikost 14 B nebo 248 B. Rozsah (délka) sítě může být až 1 000 m (ovšem mezi jednotkami 700 m). Umožňuje navíc napájet jednotky přímo po sběrnici, není tedy zapotřebí zvláštních napájecích vodičů. Plně definuje síťovou transportní a aplikační vrstvu. Jakou fyzickou vrstvu lze použít prakticky cokoliv – kroucenou dvojlinku, síťové vedení, rádiový přenos, infračervený přenos, média založená na IP (Ethernet, WiFi, BlueTooth, FireWire).

CIB Common Instalation Bus [5] – tato sběrnice je založena na technologiích firmy Teco, a. s. Umožňuje komunikovat rychlostí 19,2 kb/s s odezvou do 150 ms při plném zatížení. Současně s daty se přenáší také napájení pro připojené jednotky. Tím pádem se minimalizuje počet vodičů nutných pro napájení. Jediné, co je nutné dodržet je správná polarita 24 V napájecího okruhu a vyvarovat se kruhu v topologii. Na jednu větev může být připojeno až 32 jednotek, je-li potřeba více, lze síť rozšiřovat pomocí dalších master modulů.

2 Linux a embedded systémy

Tato část popisuje možnosti pro výběr Linuxové platformy a dále nastiňuje vlastnosti a vztah Linuxu k vestavěným systémům.

2.1 Vymezení pojmu Linux

Pojem Linux označuje jen jádro samotného operačního systému. Proto se někdy používá označení GNU/Linux, které vyjadřuje kompletní operační systém, kde jsou zahrnuty i části z projektu GNU. Jde především o knihovnu jazyka C (glibc),

překladač (gcc) a další. Samotný tvůrce Linuxu však jednou k tomuto tématu napsal:

> *Umm, this discussion has gone on quite long enough, thank you very much. It doesn't really matter what people call Linux, as long as credit is given where credit is due (on both sides). Personally, I'll very much continue to call it "Linux", but there have already been distributions that call it "Linux Pro(tm)" etc, which I don't find all that surprising.*
>
> – Linus Torvalds, příspěvek ve skupině comp.os.linux.misc

Z tohoto důvodu budeme dále používat nejen pro jádro, ale i pro celý systém prosté označení Linux.

2.2 Výběr platformy

Výběr vhodné platformy je důležitou částí při tvorbě jakéhokoli řídicího systému. Po výběru architektury a rodiny procesoru musíme vybrat, do jaké míry budeme vlastní hardware tvořit. Možností je několik:

- Použít existující řešení
- Využít CoM (Computer on Module) a vytvořit vlastní nosnou desku
- Vytvořit kompletně vlastní řešení

Použití existujícího řešení přináší mnoho výhod. Počínaje hotovým a funkčním návrhem, který je často otestovaný a ověřený komunitou uživatelů, přes možnost ihned objednat funkční desky. Oproti tomu je vývojář limitován životností a velikostí daného produktu. Navíc je potřeba vycházet z toho, jak byl produkt připraven (omezení vstupně výstupních periferií apod.). Někteří výrobci přímo podporují některé operační systémy, proto může být při tvorbě výsledného systému věnován čas jen přípravě samotné aplikace. Mezi zástupce patří například deska BeagleBoard[1].

Na pomyslném středu stojí využití CoM (někdy také označováno SoM – System on Module). Jde o menší desku, která typicky obsahuje procesor, operační paměť a paměť typu flash pro uložení dat. Při tvorbě systému pak vytváříme jen nosnou desku, do které se CoM vkládá. Výhodou tohoto přístupu je, že tvorba nosné desky je v porovnání s tvorbou celé platformy mnohem jednodušší. Odpadá také jedna z nejnáročnějších částí náchylných na chyby – propojení procesoru s pamětmi. Mezi zástupce patří například moduly firmy Toradex[2].

[1] Domovská stránka: http://beagleboard.org/.

[2] Více viz http://www.toradex.com/Products/Colibri-Computer-On-Module.

Návrh vlastního řešení přináší možnost vytvořit zařízení téměř libovolného rozměru spolu s možností využití potřebných periferií, vždy ale podle možností vybraného procesoru a okolních komponent. Životnost produktu je limitována životností jednotlivých komponent a také je snížena cena výsledného řešení. Oproti použití existujícího řešení zde ale přichází zásadní nevýhody v podobě vývoje a testování celého zařízení. Při návrhu vlastního řešení je možné a dokonce vítané využít znalostí z návrhu různých vývojových kitů, které existují vždy pro dané rodiny procesorů. Dokumentace k vývojovým kitům často obsahuje řešení problému z errata dokumentů a mnohdy se jedná opět o otestované zapojení, včetně podpory menší komunity ohledně konfigurace systému.

V našem případě došlo k výběru návrhu vlastního řešení. Ačkoliv toto řešení přináší víc práce, testování, problémů a delšího vývoje, potřeba svého vlastního zařízení o konkrétním rozměru, menší ceně a nezávislosti na výrobci zvítězila.

2.3 Historie

První verze Linuxu byla vydána v létě roku 1991 finským vývojářem Linusem Torvaldsem. Původně šlo o OS vytvořený ve volném čase. V devadesátých letech pomohl rozvoj Internetu a komunit kolem Linuxu k vytvoření prvních distribucí. Během několika let se tento univerzální operační systém vyvinul tak, že je možné jej použít v široké škále zařízení – od vestavěných systémů, přes pracovní stanice až k vysoce výkonným serverům.

2.4 Použití ve vestavěných systémech

Linux má několik vlastností, které jsou vhodné při nasazení ve vestavěném systému [6].

První může být kvalita a spolehlivost kódu. Ač jde o těžko měřitelnou veličinu, je několik faktorů, podle kterých ji můžeme porovnávat. Kód jádra je modulární s přesně definovanou strukturou. Díky tomu je možné snadno přidávat nebo odebírat různé funkce. Při znalosti principů OS je také kód jádra dobře čitelný. Z pohledu stability je jádro prověřeno mimo jiné díky velkému množství nasazení v „kritických“ aplikacích (např. servery).

Další vlastností, která plyne z vývoje s otevřeným zdrojovým kódem, je dostupnost zdrojových kódů a to, že Linux nepatří žádné firmě. Zdrojové kódy jsou vždy dostupné na Internetu – a to bez poplatku. Licence, pod kterou jsou vydávány, je GNU GPLv2. Ta umožňuje svobodné šíření a použití za předpokladu, že všechny změny jsou opět „vráceny“ komunitě. Výhodou je, že se do výsledné ceny produktu licence za software vůbec nemusí promítnout.

Na trhu existuje několik firem, které nabízí podporu pro tento OS. Nabízí konzultace až vývoj ovladačů. Je také možné se při vývoji spolehnout na pomoc komunity, která ale není zaručena a nejde vymáhat.

V souvislosti s komunitou a samotným rozšířením Linuxu najdeme další silnou stránku. Jde o podporu hardware, síťových protokolů a dostupné utility. Někteří výrobci procesorů už přímo podporují Linux a uvolňují vlastní podporu. Vždy se však objeví nějaký hardware, pro který nemusí být ovladač dostupný. Pro vestavěné systémy, které potřebují pracovat s počítačovou sítí, nabízí Linux velkou podporu ze strany síťové vrstvy.

3 Příprava Linuxového systému

Pro úspěšný provoz Linuxového systému je potřeba provést několik kroků. Na začátek je potřeba zvolit způsob propojení vývojového stroje a cílové platformy. Následně vytvořit vývojové prostředí, ve kterém bude možné překládat programy pro cílovou platformu. Dále je potřeba vybrat části jádra, které budou přeloženy. Následuje část tvorby obsahu souborového systému. V neposlední řadě je potřeba připravit software pro zavedení systému (bootloader).

3.1 Propojení s cílovou platformou

Pro vývoj je možné použít počítač s operačním systémem Linux i Microsoft Windows. V případě Linuxu je však vývoj značně ulehčen. Většina utilit a vývojových nástrojů je určena právě pro tento operační systém. Jedinou oblastí, pro které bývá podpora pro OS Windows větší, jsou různé hardwarové nástroje (debuggery, programovací kabely). Pro většinu však existuje alespoň neoficiální podpora ze strany linuxové komunity.

Při vývoji je potřeba zvolit způsob komunikace mezi cílovou platformou a vývojovým strojem. Existují tři způsoby.

3.1.1 Přímé propojení

V tomto případě je cílová platforma s vývojovým strojem propojena fyzicky pomocí kabelu. Může jít např. o sériovou linku nebo ethernet. Bootloader, jádro i samotný souborový systém jsou přenášeny pomocí tohoto kabelu. Někdy je možné zároveň pomocí tohoto propoje provádět ladění. Výhodou je, že změny provedené v souborovém systému jsou hned dostupné na cílové platformě. K propojení se používají typicky protokoly TFTP (pro přenos jádra) a NFS (pro souborový systém).

3.1.2 Vyměnitelné médium

K přenosu vytvořených obrazů jádra a souborového systému se v tomto případě používá vyměnitelné médium. V případě cíle může jít např. o paměťovou kartu. Ve srovnání s předchozím způsobem je přenos výsledků pomalejší. Na druhou

stranu je cílová platforma při běhu nezávislá na vývojovém stroji, což odpovídá výslednému nasazení.

3.1.3 Přímý vývoj

Vývoj lze provádět přímo na cílové platformě. Je možné vytvořit překladač a všechny potřebné nástroje na cílové platformě. Cílová platforma však musí být dostatečně výkonná a vstup/výstup musí umožňovat vývojářům práci. Výhodou je, že není potřeba vytvářet žádné cross-platform (viz dále) nástroje, protože vývoj probíhá přímo na cílové platformě.

3.2 Vývojové prostředí

V případě vývoje na odlišných architekturách (což je typické pro vývoj vestavěných systémů), je potřeba vytvořit tzv. cross-platform nástroje, kde jde především o překladač. Dále je potřeba vybrat a přeložit knihovnu jazyka C.

Existuje několik nástrojů, které tuto práci automatizují a jsou dostupné zdarma. Kromě tvorby překladače se zároveň starají o překlad jádra, programů a vytvoření výsledného systému souborů. Mezi tyto nástroje patří např. Ptxdist[3], Buildroot[4], OpenEmbedded[5], LTIB[6] a ELDK[7]. Jednoznačnou výhodou těchto nástrojů je urychlení vývoje a reprodukovatelnost výsledného systému. Stačí zachovat konfiguraci pro daný nástroj a systém je možné přeložit na jiném stroji. Většina stahuje zdrojové kódy přímo z Internetu.

Jako v případě podpory pro Linux i zde existují společnosti, které se zabývají přípravou těchto nástrojů. Někdy jde o vlastní, v jiných případech rozšiřují funkčnost stávajících.

Ve světě vestavěných linuxových systémů je nejznámějším překladačem gcc (GNU C compiler). Umožňuje překládat spustitelné programy pro velké množství architektur (x86, PowerPC, MIPS, ARM, Motorola 68000, ...). Nedílnou součástí výsledného programu v jazyce C je i standardní knihovna. V jejím případě existuje několik implementací. Ty vznikly především proto, že knihovna glibc, používaná na pracovních stanicích a serverech, je příliš velká (řádově jednotky až desítky megabajtů). Místo je ale v případě vestavěných systému často velmi cenný zdroj. Kromě již zmíněné knihovny glibc[8] je možné použít knihovny uClibc[9] a Diet libc[10]. Tyto knihovny se vyznačují především malou velikostí (řá-

[3]Domovská stránka: http://www.pengutronix.de/software/ptxdist/index_en.html
[4]Domovská stránka: http://buildroot.uclibc.org/
[5]Domovská stránka: http://www.openembedded.org/index.php/Main_Page
[6]Domovská stránka: http://bitshrine.org/ltib/
[7]Domovská stránka: http://www.denx.de/wiki/ELDK-5
[8]Domovská stránka: http://www.gnu.org/s/libc/
[9]Domovská stránka: http://uclibc.org/
[10]Domovská stránka: http://www.fefe.de/dietlibc/

dově stovky kilobajtů). Na druhou stranu nepodporují všechny části standardů POSIX. Jde však většinou o části, které nejsou z hlediska vestavěných systémů podstatné. Naproti tomu existuje implementace EGLIBC[11] (Embedded glibc), která je v základním nastavení binárně kompatibilní s knihovnou glibc.

3.3 Překlad výsledného systému

Tato část se zabývá překladem jádra, programy pro systém, tvorbou systému souborů a bootloaderu. Tyto části automatizují nástroje představené v předchozí kapitole. V konfiguraci se jen zvolí, co a jak přeložit. Následně se spustí překlad a za několik desítek minut až hodin je hotov obraz systému. Při změnách nastavení trvá překlad řádově kratší dobu (není potřeba stahovat a kompilovat již vytvořené části). Vytvořený souborový systém odpovídá FHS (Filesystem Hierarchy Standard [1]).

3.3.1 BusyBox

Jedním z programů, který se často objevuje ve vestavěných systémech, je BusyBox[12]. Jde o implementaci většiny Unixových příkazů, která je při statické kompilaci s uClibc velká přibližně 500 kilobajtů. V systému je uložena jako jeden binární program. Všechny utility na ni ukazují symbolickou linkou. Právě podle tohoto odkazu program zjistí, co za utilitu má spustit. Kromě standardních utilit jako `cp`, `ifconfig` a `sleep` může být do překladu zahrnut i jednoduchý webový nebo FTP server.

3.3.2 Bootloader

Nedílnou součástí vestavěného systému s operačním systémem Linux je bootloader. Těch bývá při startu systému v řadě více za sebou. Start systému pak může probíhat následujícím způsobem: Po připojení napájení začne procesor provádět bootloader uložený v ROM procesoru (označovaný např. RBL – ROM Bootloader). Ten má za úkol načíst první blok z nevolatilní paměti (např. flash) do interní paměti RAM a spustit běh. Protože je tento blok malý a ještě není inicializována externí paměť RAM, bývá v něm uložen jednoduchý bootloader (často označovaný jako stage 1). Jeho úkolem je inicializace externí RAM a načtení dalšího, většího bootloaderu (stage 2). Ten už umožňuje zavedení jádra operačního systému.

Stage 2 bootloadery umožňují načtení jádra přes síť, přímo z pamětí flash nebo ze souborového systému na paměti flash. Bootloader také může provádět zápis do flash paměti.

[11] Domovská stránka: http://www.eglibc.org/
[12] Domovská stránka: http://www.busybox.net/

Menší bootloadery (stage 1) jsou často spojeny s konkrétním procesorem. U stage 2 bootloaderů už ale nalezneme několik implementací. Jmenujme např. volně dostupné (pod licencí GPL) zástupce U-Boot[13] a RedBoot[14].

Bootloader U-Boot nabízí možnost vytvoření tzv. SPL (Second Program Loader) bootloaderu, který umožní nahrazení stage 1 i stage 2 bootloaderu. Po nastavení a několika úpravách je možné při standardním překladu vytvořit dva programy: SPL a vlastní U-Boot. Myšlenka je taková, že samotný „velký“ U-Boot, který má velkou funkcionalitu, není při standardním startu potřeba. Proto je možné jej spustit jen v případě potřeby (např. výroba nebo servisní zásah). SPL tedy rovnou načte jádro a tím zastoupí funkci obou bootloaderů. Výsledkem je jednodušší a rychlejší start systému.

4 Paměťová zařízení a souborové systémy

Uložení dat ve vestavěných systémech se většinou liší od způsobu, jakým je provedeno na pracovních stanicích a serverech. Vestavěné systémy používají úložiště na bázi flash pamětí nebo flash disků. Tyto paměti se liší od klasických disků (velikostí, způsobem přístupu, rozdělením na oddíly, spolehlivostí, ...), proto jsou i nástroje na jejich správu odlišné.

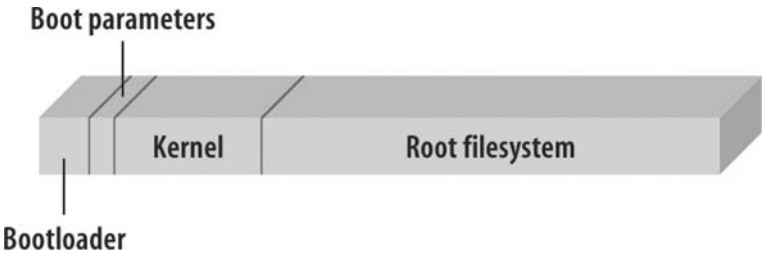
4.1 Flash paměti

U tohoto typu pamětí (NAND nebo NOR) je potřeba brát v úvahu jejich životnost. V závislosti na typu se udává počet zápisů na jeden blok v rozmezí desítek až stovek tisíc. „Klasické“ použití tohoto typu paměti a souborového systému by vedlo k rychlému zničení. Proto byly vytvořeny speciální přístupy a souborové systémy právě pro tento typ pamětí.

Na nejnižší vrstvě jde o MTD (Memory Technology Devices). Tento subsystém vytváří záznamy v adresáři `/dev` (kde se vyskytují další záznamy pro přístup k jednotlivým zařízením Linuxu). Ty se pak používají pro fyzický přístup k paměti. U těchto záznamů se objevil problém s jejich zařazením. Klasicky jsou zařízení v Linuxu rozdělena na znaková (`char`) a bloková (`block`). Flash paměť však nespadá ani do jedné kategorie. Přístup po znacích se do paměti typicky neprovádí. Na druhou stranu, bloky flash paměti jsou řádově větší než klasický blok (512 bajtů). Navíc do flash paměti není umožněn přímý zápis. Příslušný blok musí být nejprve smazán, až pak je možné do něj začít zapisovat. Výsledkem je to, že existují jak znakové, tak blokové typy zařízení. Příslušné nadřazené vrstvy pak vybírají vhodnou reprezentaci a adekvátně s ní pracují.

[13] Domovská stránka: http://www.denx.de/wiki/U-Boot
[14] Domovská stránka: http://sourceware.org/redboot/

Subsystém MTD se také stará o rozdělení paměti na jednotlivé části (oblast pro bootloader, jádro, souborový systém). Na rozdíl od rozdělení na partice, které známe z pevných disků počítačů, se paměť rozděluje na oblasti dané svým začátkem a koncem. Toto rozdělení se nezapisuje do paměti, ale je uloženo jako struktura v jádru. Definice této struktury se provádí buď přímo ve zdrojovém souboru jádra nebo se předává při startu jádra jako parametr. Typické rozdělení paměti ve vestavěním systému nalezneme na obrázku 1.



Obr. 1: Rozdělení paměti ve vestavěném systému. Převzato z [6]

Důležitou částí, která se používá u flash pamětí, je FTL (Flash Translation Layer). Jde o mezivrstvu, která se stará o rozkládání zápisů na jednotlivé bloky flash paměti. Jejím úkolem je rozložit zápisy rovnoměrně tak, aby nedocházelo k opotřebení jen jedné části flash paměti. Tuto mezivrstvu obsahují všechny USB paměti. Jde typicky o samostatný čip, který ve vestavěném systému nenajdeme – proto potřebuje speciální přístupy.

Zajímavý přístup pro vestavěné systémy nabízí eMMC. Jde o čip, který v sobě kombinuje paměťovou kartu a řadič. Paměť má rozhraní MMC, tedy zajišťuje vrstvu FTL. Vzhledem k bohatému výběru souborových systémů, které jsou určené přímo pro použití na NAND nebo NOR čipech, je ale toto řešení vhodné spíše do menších vestavěných systémů.

4.2 Souborové systémy

Vzhledem k nízkým životnostem pamětí flash a potřebám vestavěných systémů bylo potřeba vytvořit nové souborové systémy. Ty musí rovnoměrně rozkládat využívání bloků, musí být odolné vůči výpadku napětí a musí být rychlé.

Na jedné straně existuje část souborového systému, která je neměnná po celou dobu života vestavěného systému. Pro tyto části je možné použít souborový systém, který je pouze pro čtení. Jejich výhodou je jednoduchost, rychlost a odolnost proti výpadku. Zástupci z této rodiny jsou např. Cramfs a SquashFS (oba zároveň podporují kompresi dat).

Na druhou stranu jsou případy, kdy potřebujeme neustále ukládat nebo přepisovat data. I nejlepší souborové systémy pro flash paměti by po čase způsobily jejich zničení. Proto se používají souborové systémy v paměti RAM. Ty sice ztrácí obsah při odpojení napájení a mezi restarty systému, na druhou stranu vůbec neničí flash paměť. Používají se typicky pro adresář `/tmp`, jehož velikost bývá malá a není potřeba uchovávat obsah mezi restarty.

Poslední skupinou jsou souborové systémy, které umožňují zápis a zároveň pracují přímo na paměti flash. Jmenujme několik zástupců, např. JFFS2, UBIFS nebo YAFFS. Vzhledem k odlišným přístupům popíšeme princip prvních dvou zmiňovaných.

JFFS2 pracuje na principu logování. Začíná na začátku paměti a každou změnu zapisuje postupně ke konci flash paměti. V případě, že dojde paměť, spustí se tzv. garbage collector. Ten projde místa, kde jsou označeny smazané soubory. V jejich místech smaže celý blok, takže je znovu použitelný. Nevýhodou je pomalejší připojení (mount) souborového systému. Musí se procházet kvůli rekonstrukci souborů.

UBIFS je souborový systém, který je postaven na mezivrstvě UBI, která se stará o mapování fyzických bloků na logické bloky. Má rezervu několika fyzických bloků. V případě, že dojde ke zničení fyzického bloku, je jeho logický blok přesunut na jiný fyzický blok. Tato operace je transparentní pro UBIFS, proto je tento souborový systém ve výsledku jednodušší. Oproti JFFS2 je UBIFS tzv. write-back souborový systém. Změny jsou na disk ukládány z cache, až je to opravdu nutné. Mezi další možnosti UBIFS patří komprese dat.

5 Návrh a implementace centrální jednotky

Ze strany software byl pro centrální jednotku požadavek, aby využívala Linux. Z pohledu hardware vylo vyžadováno připojení do LAN, úložiště pro provozní data a dostupnost několika sériových portů. Díky zkušenostem s procesory od společnosti Texas Instruments byl vybrán procesor AM1707. Další součásti byly záležitostí dodacích podmínek distributorů a zkušeností s daným výrobcem.

Procesor AM1707 je postaven na starším jádře ARM926EJ-S, obsahuje 2 rozhraní pro přístup k paměti (jedno primárně pro nevolatilní paměť, druhé pouze pro volatilní paměť), 3 sériové linky, ethernet, SPI a mnoho další periférií.

5.1 Rozvržení systému

Centrální jednotka je rozdělena do 3 částí (modulů). Všechny moduly jsou propojeny pomocí pinheaderů.

První modul obsahuje napájecí část celého systému a rozhraní na speciální sběrnici elektroinstalace. Tato sběrnice je převedena na sériovou linku a přivedena do druhého modulu.

Druhý modul je „mozkem“ celého systému. Obsahuje již dříve zmíněný procesor AM1707, NAND paměť o velikosti 2 Gb s 16 bitovým přístupem, 128 MB SDRAM paměť se 32 bitovým datovým přístupem, rozhraní pro ethernet, malou flash paměť pro uchovávání konfigurací a programovatelné rozšiřující rozhraní realizované pomocí mikrokontroléru STM32F100. Dále různé podpůrné obvody, zapojení a konektory, které umožňují přístup z dalších modulů.

Třetí a poslední modul vytváří určité rozhraní směrem k uživateli. Obsahuje malý LCD displej, jednoduchou klávesnici, RF rozhraní, konektor pro ethernet, slot pro paměťovou kartu, servisní rozhraní a jednoduchou signalizaci.

Všechny tři moduly jsou do sebe zapojeny jako „sendvič“.

5.2 Chyby v návrhu

Při návrhu se počítalo se zapojením obou sběrnic pro připojení pamětí. Jedna pro NAND flash a druhá sběrnice pro SDRAM. Současně bylo plánováno využít i rozhraní pro SD kartu, které procesor AM1707 nabízí. Vodiče pro přímý přístup na SD/MMC kartu jsou sdíleny s vodiči pro NAND paměť, nicméně na procesoru je výstup pro chip select, který je možné využít pro multiplexor (řídí, jestli se využívá NAND nebo paměťová karta).

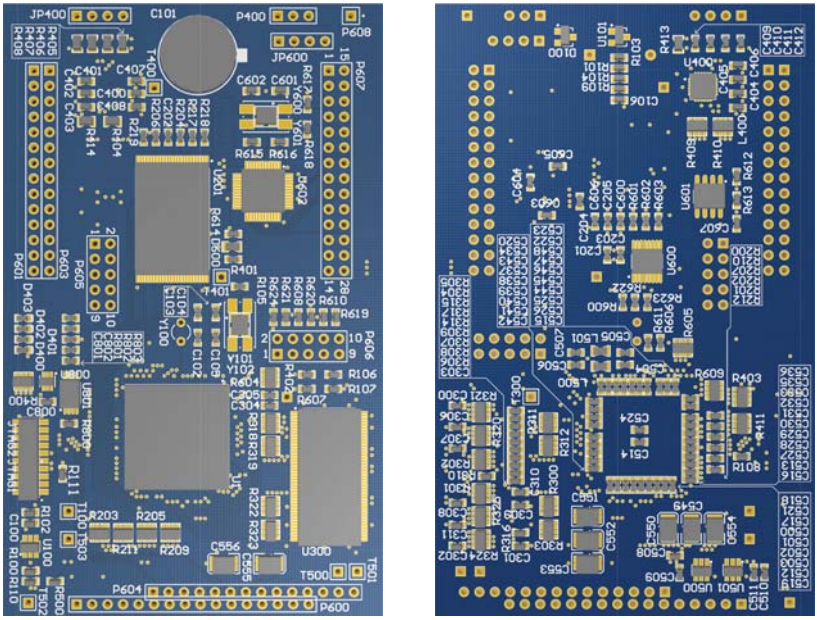
Procesor tedy sice obsahuje obě potřebná rozhraní, neumožňuje ale jejich sdílený běh. Při konfiguraci jádra je nutné si vybrat, které zařízení bude využito a po celý běh je potřeba přistupovat pouze k tomuto typu zařízení. Tuto limitaci je možné obejít, ale zásah do Linuxu by byl velký. Bylo by nutné vždy při zápisu na NAND provést přemapování pinů na modul, který obsluhuje NAND a následně provést zápis. Stejná akce by musela být provedena při zápisu na paměťovou kartu. Zároveň by musely být doplněny zámky, které by hlídaly přístup.

Kvůli těmto obtížím a tomu, že NAND paměť je pro zařízení kritická, rozhodli jsme se využít paměťové rozhraní pouze pro NAND paměť a paměťovou kartu připojit pomocí rozhraní SPI.

5.3 Oživení desky plošného spoje

Zapojení centrální jednotky bylo inspirováno vývojovým kitem k danému procesoru. Texas Instruments, jakožto výrobce čipu, má navíc přímou podporu pro U-Boot a Linux na jeho vývojovém kitu. Přestože byla většina komponent odlišná a využito bylo jen několik periférií, vychází podpora pro námi vytvořenou jednotku z podpory vývojového kitu.

Na obrázku 2 je vizualizace vytvořené desky.



Horní strana Spodní strana

Obr. 2: Vizualizace desky

Vytvořená deska má rozměry 101 mm × 62,5 mm. Skládá se z šesti vrstev a je vytvořena ve třídě přesnosti VII.

Při oživování bylo využito pro základní testy rozhraní JTAG, přes které byly nahrávány jednoduché programy, které testovaly funkčnost periférií. Po úspěšném provedení testů přišla další fáze – portace bootloaderu a Linuxu. Ta spočívala především v úpravách nebo doplňování definic specifických pro naši desku.

5.4 Portace software a vývoj

Jako vývojový nástroj pro tvorbu překladače, knihovny jazyka C a dalších komponent výsledného systému byl využit dříve zmiňovaný Buildroot. Jde o jednoduchý nástroj založený na klasických makefilech. Pro vytvoření malého Linuxového systému je plně dostačující. V úvahu připadaly i jiné nástroje, např. OpenEmbedded, ale ten je vhodný u větších a složitějších systémů. Umožňuje totiž mimo jiné i tvorbu balíčků, které usnadňují další správu a úpravy software vestavěného systému. V našem případě však tuto možnost nevyužijeme.

Procesor AM1707 umožňuje natažení spouštěného programu i přes sériovou linku. Jde o jeden z několika boot režimů, které nabízí. Díky tomuto způsobu

může být vývoj prováděn i bez rozhraní JTAG. Zároveň jde o způsob, kterým může být nahrán první software do desky. Přes sériovou linku se nahraje malý program, který si z počítače vyžádá bootloader (také přes sériovou linku) a ten nahraje do paměti. Pak spustí právě nahraný bootloader, který se při prvním spuštění postará o natažení jádra a souborového systému. Tento větší objem dat je ale přenášen díky možnostem bootloaderu přes síť – natažení tak velkého objemu dat by přes sériovou linku zabralo několik jednotek až desítek minut.

Díky přítomnosti síťového rozhraní bylo také možné rychle a efektivně ladit Linux. Jádro bylo vždy přeneseno přes síť a okamžitě z bootloaderu spuštěno, nebylo tedy potřeba nic ukládat do flash paměti – tím nedocházelo k jejímu zbytečnému opotřebení.

5.4.1 Podpora AIS

Z pohledu rychlého startu a jednoduchosti bootloaderů je jednou z výhod procesoru AM1707 podpora protokolu AIS (Application Image Script). Tento protokol najdeme i v jiných procesorech firmy Texas Instruments. V ROM procesoru je nahrán jednoduchý bootloader, který umí zpracovat několik základních povelů. Jde především o zápis dat do paměti RAM. Pokud najde ROM bootloader při startu magickou konstantu, začne podle jednoduchých instrukcí provádět program.

Příprava probíhá tak, že je na počítači po přeložení bootloaderu U-Boot programem AISgen vygenerován binární soubor s potřebnými instrukcemi. Kromě samotného bootloaderu je do binárního souboru připojena definice registrů periférií mikroprocesoru. Tento binární soubor je nahrán do NAND paměti.

Při startu procesoru je v paměti NAND nalezena magická konstanta, takže pokračuje vykonávání instrukcí. V první řadě jsou na příslušná místa překopírovány obsahy registrů periférií, čímž dojde k jejich inicializaci. Dále jsou v binárním souboru AIS nalezeny instrukce, které říkají, že další bloky paměti NAND mají být překopírovány do RAM. Zde dojde k překopírování bootloaderu U-Boot. Poslední instrukce v souboru AIS říká, že se má začít vykonávat právě natažený U-Boot v paměti RAM (skočí se na první adresu a začne vykonávání programu). Tato metoda je velmi rychlá a umožňuje nám úplnou eliminaci stage 1 bootloaderu.

6 Závěr

Práce shrnuje základní popis inteligentních domů, dále přechází k obecnému popisu použití Linuxu ve vestavěných systémech a následně popisuje zkušenosti při tvorbě centrální jednotky inteligentního domu. Jde samozřejmě jen o jeden ze způsobů řízení. Inteligentní domy, případně i budovy, se začínají stále více

prosazovat, takže se v budoucnu dočkáme dalších přístupu a rozšíření technologií v této oblasti.

Poděkování

Tento výzkum a vývoj byl financován v rámci projektů „Výzkum informačních technologií z hlediska bezpečnosti“ – MSM0021630528 (ČR), „Pokročilé bezpečné, spolehlivé a adaptivní IT“ – FIT-S-11-1 (ČR) a „Centrum excelence IT4Innovations“ – IT4I-CZ 1.05/1.1.00/02.0070 (ČR).

Literatura

[1] *Filesystem Hierarchy Standard*. Filesystem Hierarchy Standard Group. 2004. Dostupné z: http://www.pathname.com/fhs/pub/fhs-2.3.pdf

[2] Vojáček, A.: *Sběrnice LonWorks*. [online]. 2005 [cit. 2012–09–14]. Dostupné z: http://automatizace.hw.cz/clanek/2005040501

[3] Vojáček, A.: *Úvod do BACnetu – Building Automation and Controls Network*. [online] 2012 [cit. 2012–09–14]. Dostupné z: http://automatizace.hw.cz/uvod-do-bacnetu-building-automation-and-controls-network

[4] Vojáček, A.: *Sběrnice KNX pro řízení budov*. [online]. 2006 [cit. 2012–09–14]. Dostupné z: http://automatizace.hw.cz/clanek/2006061001

[5] Klaban, J.: *Inels a sběrnice CIB – moderní systém inteligentní elektroinstalace*. [online]. 2008 [cit. 2012–09–14]. Dostupné z: http://www.odbornecasopisy.cz/index.php?id_document=38218

[6] Yaghmour, K., et al. *Building Embedded Linux Systems*. 2. vydání. USA : O'Reilly Media, Inc., 2008. 442 s. ISBN 978-0-596-52968-0.

SmartGrid & Smart metering

Radek Semrád

E-mail: radek.semrad@eon.com

1 Úvod

Struktura celého energetického systému se rychle mění. Velmi rychlý vývoj především ve výrobě na poli tak zvaného využívání obnovitelných zdrojů energie a zároveň stárnoucí současná infrastruktura vytváří nové nároky na rozvoj přenosových a distribučních soustav po celé Evropě. Směr, kterým se nová strategie ubírá je také nastaven rozvíjející se legislativou, otevíráním energetického trhu a potřebou investic. Principy, kterými se řídil energetický průmysl po dobu téměř sta let přestávají splňovat dnešní požadavky. Dnešní centralizované sítě se musí stát pružnými a adaptabilními a budou součástí interakce mezi jednotlivými částmi celého systému, které musí být vysoce citlivé na výkyvy v poptávce a dostupnosti. Snahou tohoto příspěvku je vedle stručného vysvětlení problematiky SG,SMR, hlavně popis současného stavu v České Republice v kontextu distribučních soustav energie celé Evropy. Je potřeba předeslat, že Česká republika si nestojí vůbec špatně

2 Nové technologie využívání energetických sítí

V posledních pěti letech se zásadně změnilo použití energetických sítí.
Lze pozorovat:

- Nové rozložení zdrojů a odběratelů

  Díky novým technologiím pro výrobu elektrické energie se mění stávající, převážně centralizovaná distribuční infrastruktura, na stále více decentralizovanou s přibývajícím počtem středních a malých výroben využívajících energii slunce, větru, vody či jiných (kogenerační jednotky, spalovny jiných než fosilních paliv, ...). Využití těchto zdrojů dává odběratelům možnost stát se zároveň dodavateli do distribuční sítě. Zároveň vznikají nové typy odběrných míst např. dobíjecí stanice pro elektromobily

- Využití Smart Grid

  Smart Grid je název nové koncepce uplatňující se v rozvodu silové elektrické energie a plynu. Myšlenka spočívá v převodu stávající pasivní distribuční sítě na tzv. „chytrou“ síť, nebo-li distribuční síť doplněnou o aktivní prvky propojené komunikační sítí. Jedná se v podstatě o rozložení regulačních mechanismů sítě směrem k přenosovým soustavám a koncovým zákazníků, tedy na odběrná místa představované měřidlem. Vzniká tak parciální část systému nazývaná Smart metering.

- Využití Smart Metering

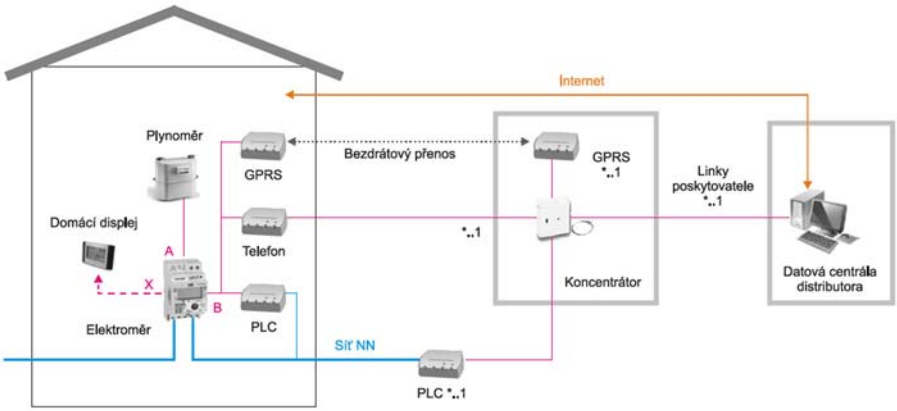
  Myšlenka Smart Metering nebo také Smart Meter Reading (SMR) spočívá v nahrazení klasických pasivních měřidel enegie u konečného spotřebitele aktivními sofistikovanými měřidly, která jsou schopná měřit nejenom fyzikální veličiny týkající se spotřeby a kvality dodávané energie, ale zároveň jsou aktivními prvky v distribuční síti, které odesílají informace o průběhu a stavu jednotlivých veličin a zároveň přijímají řídící signály kterými lze měřidlo uvést do požadovaného stavu či ho přeprogramovat. V současné době probíhají pilotní projekty nasazení těchto systémů po celé Evropě.

- Nové požadavky trhu na distributory (OTE – Operátor trhu z energií Regulační úřad ČR)

  Ekonomický tlak vytvářený rostoucí cenou energií způsobuje snahu zkrátit dobu od spotřeby k fakturaci. Rovněž burzy z energií konzumují čím dál tím přesnější a rychlejší data. Dnes je vyžadováno měření příkonu po 15 minutách a fakturace jednou měsíčně. To lze dosáhnout jen nasazením měřidel s telekomunikačními moduly. Trh vyžaduje i stabilitu sítě a to lze dosáhnou jen řízením příkonu u spotřebitele, tedy nasazení Smart Metering systémů..

3 Smart Metering

Jak vypadá v praxi vlastní systém SMR. Předně na odběrném místě jsou měřidla s komunikačními kanály. Obvykle se volí komunikační cesta nejvýhodnější pro dané místo. Nejčastěji jde o širokopásmový přenos po silových vodičích (PLC). Zpětná vazba zákazníka pes internet slouží více méně jen ke konzultaci historických dat a utracených peněz. Zde bývá zpoždění asi 24 hodin. To je způsobeno nočním zpracováním dat v informačním systému distributora.



4 Požadavky EU na zavádění SM

Požadavky na zavádění a podobu systému inteligentního měření pro oblast elektřiny vycházejí zejména ze Směrnice evropského parlamentu a rady 2009/72/ES, ze dne 13. července 2009, o společných pravidlech pro vnitřní trh s elektřinou a o zrušení směrnice 2003/54/ES.

Směrnice předpokládá zavedení systému inteligentního měření a to zejména za účelem lepší informovanosti spotřebitelů o množství spotřebovaného média a platbách za tuto spotřebu. Při lepší informovanosti je očekáváno snížení spotřeby, což je implicitně deklarovaným cílem zavedení systému inteligentního měření.

Směrnice v příloze I stanoví, že
„Členské státy zajistí zavedení inteligentních měřicích systémů, které podpoří aktivní účast spotřebitelů na trhu s dodávkami elektřiny. Zavedení těchto měřicích systémů může být podmíněno ekonomickým posouzením všech dlouhodobých nákladů a přínosů pro trh a jednotlivého spotřebitele nebo posouzením toho, jaký způsob inteligentního měření je z hospodářského hlediska nejpřiměřenější a nákladově nejefektivnější a jaký harmonogram jejich distribuce je proveditelný.

Toto posouzení se provede do dne 3. září 2012.
Na základě tohoto posouzení členské státy nebo jakýkoli příslušný orgán jimi určený připraví rozvrh, jehož cílem je zavedení inteligentních měřicích systémů do 10 let.“

Jak vyplývá ze stránek Ministerstva Průmyslu České Republiky zatím jsme nasazení SM pravděpodobně odložili. Odpověď z EU ještě nedošla.

5 Zahraniční zkušenosti se zaváděním SM

Zkušenosti s reálným zavedením inteligentního měření má v současné době Kanada, Itálie, Švédsko. Další uváděné země v době zpracování této Koncepce jsou ve fázi vyhodnocování pilotních projektů, spouštění přípravných fází, nebo vůbec ještě nezačaly a připravují národní studie (ekonomické posouzení).

Na základě získaných informací uvedených v této studii o zavedení inteligentního měření v jednotlivých státech EU lze motivy k zavedení SM rozdělit do dvou skupin:

- Politické (Švédsko, Norsko)
  - Liberalizace trhu s energií (Itálie, Švédsko, USA)
- Ekonomické
  - Dosažení úspor energie (UK, Nizozemí, USA, Kanada)
  - Řízení spotřeby prostřednictvím snížení špiček spotřeby (Itálie, Švédsko,
  - Kanada)
  - Eliminace netechnických ztrát (Itálie, UK)
  - Přesné účtování a tím související pokles reklamací (Itálie)
  - Zvýšení kvality služeb pro zákazníky (UK)
  - Snížení nákladů na odečty (Itálie)

V jednotlivých zemích se uvedené motivy prolínají.

Jednotlivé země mají jinou motivaci, jiné způsoby řešení, rozdílné technologie, jiné hranice odpovědností jednotlivých subjektů, a to je důvodem, že se jejich přístupy k problematice SM liší. Zmapované země dosáhly různých výsledků v oblasti zavádění SM, které jsou velice těžko porovnatelné.

6 Stav v České Republice

V ČR jsou oproti analyzovaným zemím jiné výchozí podmínky. Je využíván dvoutarifní produkt pro vytápění a ohřev vody, v praxi funguje účinné a spolehlivé řízení spotřeby (vyhlazení špiček) pomocí HDO. Díky fungujícímu systému zálohových plateb se nevyskytuje zásadní problém s řízením neplatičů, netechnické ztráty jsou na nízké úrovni. Je plně funkční systém operátora trhu. Lze tedy konstatovat, že významnou část přínosů, které vedou jiné státy k zavedení SM, již jsou v ČR k dispozici a účastníci trhu je aktivně využívají. Pro zhodnocení situace v souvislosti s případným zavedením systému inteligentního měření je třeba vzít v úvahu následující faktory:

- **Současnou podobu měření, řízení** a komunikace v oblastech distribuce elektřiny v České republice je možno ve srovnání s okolními zeměmi a zejména ve srovnání s průměrem EU označit za nadstandardní. Distribuce elektřiny v ČR vykazuje:

- **nízkou poruchovost** – vysokou provozní spolehlivost na úrovni, kterou bez nadměrných investic nelze dále navyšovat,
- **nízkou míru zneužívání distribučních sítí** – nízkou míru netechnických ztrát,
- **vysokou schopnost regulace zatížení** na denní úrovni při distribuci elektřiny – pomocí dnešního systému HDO, který je v prostředí EU svým rozsahem nasazení výjimečný a je plně funkční,vysokou míru automatizace distribučních soustav elektřiny od napěťové úrovně VN a výše.
- **Aktuálně dostupné technologie** inteligentního měření včetně komunikace nejsou zatím připraveny k plošnému nasazení z pohledu provozovatelnosti a spolehlivosti systému.
- **Systém HDO** umožňuje dosáhnout vyšší přenosové rychlosti a odezvy

7 Závěr

Nasazrení Smart Metering je nutností. Otázkou zůstává kdy a za jakou cenu. Základní investice je asi 800 Kč na jedno odběrné místo. Tedy přes 2,4 miliardy Kč na celou republiku plus náklay na dodatečné systémy. Z uvedeného je vidět, že je rozhodnutí na parlamentu a vládě.

Zahraniční zkušenosti se zaváděním SM

Zkušenosti s reálným zavedením inteligentního měření má v současné době Kanada, Itálie, Švédsko. Další uváděné země v době zpracování této Koncepce jsou ve fázi vyhodnocování pilotních projektů, spouštění přípravných fází, nebo vůbec ještě nezačaly a připravují národní studie (ekonomické posouzení).

Na základě získaných informací uvedených v této studii o zavedení inteligentního měření v jednotlivých státech EU lze motivy k zavedení SM rozdělit do dvou skupin:

- Politické (Švédsko, Norsko)
  - Liberalizace trhu s energií (Itálie, Švédsko, USA)
- Ekonomické
  - Dosažení úspor energie (UK, Nizozemí, USA, Kanada)
  - Řízení spotřeby prostřednictvím snížení špiček spotřeby (Itálie, Švédsko, Kanada)
  - Eliminace netechnických ztrát (Itálie, UK)

- Přesné účtování a tím související pokles reklamací (Itálie)
- Zvýšení kvality služeb pro zákazníky (UK)
- Snížení nákladů na odečty (Itálie)

V jednotlivých zemích se uvedené motivy prolínají.

Jednotlivé země mají jinou motivaci, jiné způsoby řešení, rozdílné technologie, jiné hranice odpovědností jednotlivých subjektů, a to je důvodem, že se jejich přístupy k problematice SM liší. Zmapované země dosáhly různých výsledků v oblasti zavádění SM, které jsou velice těžko porovnatelné.

Zhodnocení současného stavu v ČR z hlediska požadavků na zavádění SM

V ČR jsou oproti analyzovaným zemím jiné výchozí podmínky. Je využíván dvoutarifní produkt pro vytápění a ohřev vody, v praxi funguje účinné a spolehlivé řízení spotřeby (vyhlazení špiček) pomocí HDO. Díky fungujícímu systému zálohových plateb se nevyskytuje zásadní problém s řízením neplatičů, netechnické ztráty jsou na nízké úrovni. Je plně funkční systém operátora trhu. Lze tedy konstatovat, že významnou část přínosů, které vedou jiné státy k zavedení SM, již jsou v ČR k dispozici a účastníci trhu je aktivně využívají. Pro zhodnocení situace v souvislosti s případným zavedením systému inteligentního měření je třeba vzít v úvahu následující faktory:

- Současnou podobu měření, řízení a komunikace v oblastech distribuce elektřiny v České republice je možno ve srovnání s okolními zeměmi a zejména ve srovnání s průměrem EU označit za nadstandardní. Distribuce elektřiny v ČR vykazuje:
  - nízkou poruchovost – vysokou provozní spolehlivost na úrovni, kterou bez nadměrných investic nelze dále navyšovat,
  - nízkou míru zneužívání distribučních sítí – nízkou míru netechnických ztrát,
  - vysokou schopnost regulace zatížení na denní úrovni při distribuci elektřiny – pomocí dnešního systému HDO, který je v prostředí EU svým rozsahem nasazení výjimečný a je plně funkční,
  - vysokou míru automatizace distribučních soustav elektřiny od napěťové úrovně VN a výše.
- Aktuálně dostupné technologie inteligentního měření včetně komunikace nejsou zatím připraveny k plošnému nasazení z pohledu provozovatelnosti a spolehlivosti systému.
- Systém HDO umožňuje dosáhnout vyšší přenosové rychlosti a odezvy

Managing chytrých měřicích systémů

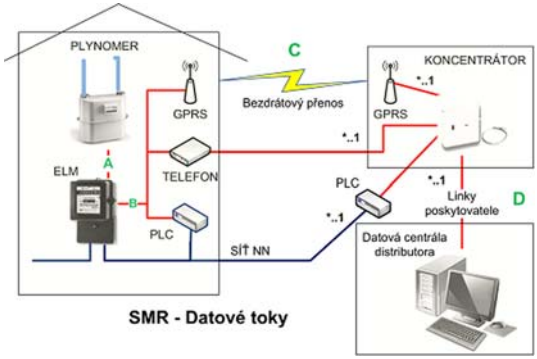
Václav Novák

E-mail: vaclav.n@iol.cz

1 Úvod

V poslední době začínají vystupovat do popředí z hlediska bezpečnosti i datové systémy užívané v energetice a plynárenství. Vedle „klasické“ ochrany energetických zařízení je potřeba zabezpečit i toky dat mezi jednotlivými přístroji. V energetice a plynárenství se začínají uplatňovat myšlenky inteligentního řízení, rozvodu a spotřeby energie nazývané „Smart Grid“. Součástí tohoto řízení je i inteligentní měření. Měřidlo je ve většině případů umístěno přímo u spotřebitele. Inteligentní měření je obohaceno o řízení měřidel a je nově nazýváno „SMR – Smart Meter Reading“. Z hlediska datových přenosů jde o velmi podstatné toky dat z centra, datové centrály, směrem k měřidlům a i naopak. Řešením bezpečnosti přenosu těchto dat je nutno věnovat velkou pozornost. Zatím na území České republiky (i EU) jsou měřidla elektřiny a plynu pouze schopny poskytovat data o dodávce či spotřebě za posledních 40–90 dnů. Pomocí nástrojů dálkových odečtů (RMR – Remote Meter Reading) jsou přenášeny do datové centrály. Zabezpečení přenosů spočívá pouze v autentizaci zdrojů a v zabezpečení dat proti pozměňování. Odečty, vzhledem k paměti přístrojů, je možné provádět opakovaně. V praxi tedy jsou útoky proti datům prakticky nulové. Navíc je možné odečítat data na místě přenosným terminálem. Jiná je situace, kdy vedle přenášení dat jsou přenášené povely k omezení nebo až k odpojení spotřebitele, což je pro inteligentní sítě nezbytné.

Tím začíná být systém velmi zajímavý z hlediska penetrace a převzetí řízení sítě nežádoucím subjektem. Prvotní je prolomení datové komunikace, ale hlavní škody mohou vznikat v oblasti rozvodu silové energie. Můžeme vyvolat celkový black-out, stejně jako odpojení celých oblastí. Obrana proti zneužití SMR je předmětem následujících úvah. Systém inteligentních měřidel nazývaný SMR - Smart Meter Reading, je pochopitelně řízen z centra, nebo-li datové centrály. Vznikají tak hlavní datové toky, které se snaží postihnout obr. 1. Jsou označeny písmeny A, B, C a D.



Obr. 1: SMR – Popis systému

2 Popis systému SMR – Smart Meter Reading

2.1 Měřidla

Na obr. 1 jsou zachyceny jednotlivé datové toky od měřidel k datové centrále a naopak. Hlavními aktéry v systému jsou pochopitelně měřidla spotřeby plynu a elektřiny. V systému SMR jsou, vedle měření, navíc schopny omezit odběr až do vypnutí a znovu zapnutí energie. Hodnoty spotřeby se zaznamenávají po 1/4 hodině, tedy za standardní den máme 96 hodnot. Jak pro plyn, tak pro elektřinu. Tyto údaje jsou uloženy v paměti přístroje až po dobu 90 dnů (ze zákona 40 dní). Tyto údaje jsou pak předmětem přenosu do datové centrály od měřidel. Samozřejmě k změněným datům se přidružují i servisní data z měřidel. Jednoduchým výpočtem získáme velikost takového bloku dat.

96 čtvrthodin $\times 6$ platných míst $\times$ 90 dní $\times$ 6 měřících kanálů $+$ 20 % dodatečných dat kanálů dostáváme zhruba 400 000 bytů surových dat z jednoho měřidla. Navíc se tato surová data přenášejí v textové podobě a jsou doprovázeny příslušnými mezerami, konci řádků apod. V budoucnu lze předpokládat tento objem dat jak pro plyn, tak pro elektřinu. Je nutné počítat s obálkami protokolů s hodnotou velikosti bloků asi 1,5 až 2 MB.

To jen pro představu o jakých objemech dat je zde hovořeno.

V opačném směru měřidla přijímají povely z datové centrály, které určují jejich chování. Může se jednat o vypnutí, zapnutí[1], omezení výkonu. V pevné paměti měřidla jsou zaznamenány i přepočtové koeficienty pro měřící prvky apod. Těmto datům říkáme „parametrizační data“. Parametrizační data je možné samozřejmě dálkově přenášet z datové centrály a tak dynamicky měnit vlastnosti měřidel. Obchodníci požadují často dynamické změny tarifů, tedy nastavení měřidel.

[1] O zapnutí zejména plynu se vedou vášnivé diskuze: Je to bezpečné? Umožňuje to legislativa? A podobně.

2.2 Koncentrátor

Dalším aktérem v našem systému je koncentrátor. To je zařízení, které z dané oblasti (vesnice, městské čtvrti apod.) sdružují a distribuují tok informací oběma směry. Běžné koncentrátory obsluhují řádově desítky až stovky měřících přístrojů. Pro bezpečnost přenosu dat nepředstavují přílišné riziko. Jsou to v podstatě jen směrovače datových bloků. V budoucnu se předpokládá analýza dat i na této úrovni pro řešení netechnických ztrát za použití topologie sítě oblasti obsluhované koncentrátorem, ale to je jiná oblast bezpečnosti.

2.3 Datová centrála

Hlavním aktérem je datová centrála. Je zdrojem příslušných povelů a zároveň přijímá doručená data a ukládá je na úložiště. Tato datová centrála je umístěna přímo v sídle firmy a podléhá firemním bezpečnostním protokolům. Obvykle je napojena na informační systém firmy přes vnitřní síť. Zde je nutno počítat z hlediska distribučních systémů jen s průniky ze strany koncentrátorů

3 Požadavky na bezpečnost

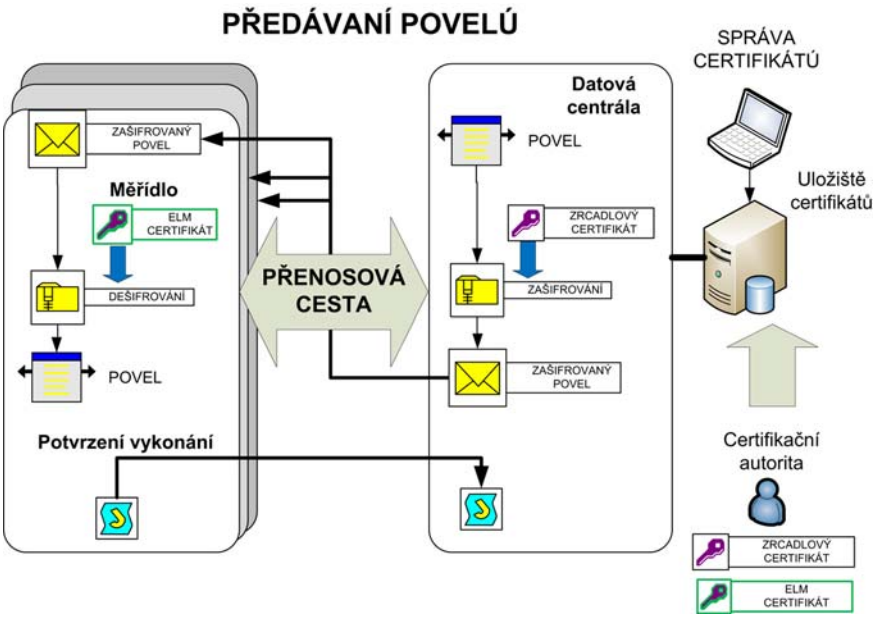
Jistě je nutné chránit všechny datové toky A, B, C a D dle obr. 1 před nežádoucími zásahy, ať co do pozměňování získaných dat, tak hlavně co do manipulace s řídícími nebo parametrizačními příkazy pro měřidla. Celá problematika zabezpečení se nám rozpadá na ochranu aktérů, tedy měřidel, koncentrátorů a datových centrál a na ochranu datových toků. Začněme s úvahami, jak zabezpečit ochranu ve směru k měřidlům. Budeme je pro jednoduchost nazývat povely.

3.1 Ochrana povelů

Samozřejmě že musíme uvažovat z hlediska bezpečnosti přenosu o šifrování. Model šifrování povelů je na obr. 2. Na obrázku chybí koncentrátor. Ten z hlediska úvah o bezpečnosti není relevantní. Připomeňme si, že slouží jen jako jakýsi distributor a shromažďovač datových bloků bez možnosti nahlížet dovnitř.

Popišme si model šifrování dle obr. 2. Povel vznikne v datové centrále. Jeho vznik je podmíněn především:
Identifikací zadávajícího operátora

- Vniknout může povel jednorázový nebo opakovatelný, pro jedno měřidlo nebo více měřidel. Rozhodnutí o míře automatizace procesu je dáno především odolností softwaru proti náhodným chybám.
- Předpokladem jsou platné certifikáty u měřidel a jejich parita se skladem certifikátů v datové centrále.



Obr. 2: Zabezpečení předávání povelů

Přidělování certifikátů měřidlům je řízeno správcem certifikátů. Správce certifikátů musí vygenerovat ve spolupráci s certifikační autoritou příslušné certifikáty pro měřidla. Správce certifikátů musí mít možnost dopravit příslušné certifikáty k měřidlům a řídit jejich životní cyklus. Stejně musí pečovat o skladiště certifikátů v místě datové centrály. Rozeslání certifikátů je možné standardně přes datovou síť. Pokud by byla nedůvěra u utilit k tomuto způsobu, je možné doručit certifikát přímo například technikem odečtů.

Povel tedy vznikne v datové centrále za výše uvedených podmínek. Před odesláním je nalezen příslušný certifikát cílového místa, kde je umístěno měřidlo. Následuje zašifrování a uložení do fronty zpráv pro odeslání. Dle možnosti přenosového média je zašifrovaný povel odeslán. Po přijetí měřidlem je příkaz rozšifrován, a pak proveden. Buďto hned nebo k danému okamžiku. Tak mohou být povely rozeslány předem a v danou vteřinu provedeny v celé oblasti. Pokud se jedná o povel s potvrzením, provede po rozhodném okamžiku datová centrála dotaz, zda byl proveden příkaz. Zde je možno uvažovat ještě před provedením příkazu o jeho odvolání apod.

Z principu vyplývá, že aktivním prvkem je datová centrála. Nepředpokládáme jakoukoli vlastní aktivitu měřidel. Z hlediska bezpečnosti je možná i varianta s aktivními měřidly. Rozhodnutí musí provést architekt softwarového řešení společně s uživatelem.

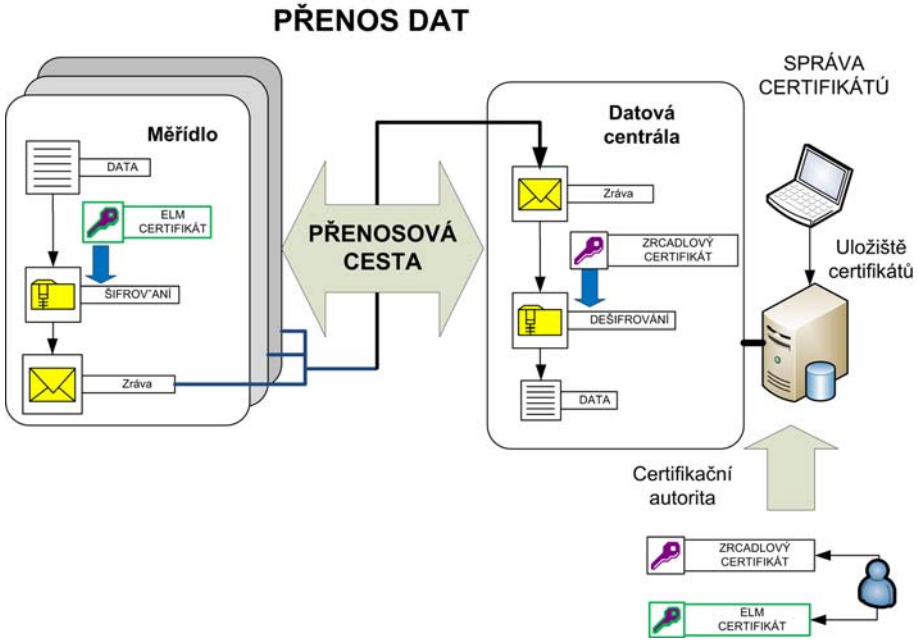
3.2 Ochrana měřených dat

Zde není nutno postupovat tak striktně, jako u povelů. Samozřejmě je zde nutno položit velký důraz na autorizaci odesílajícího měřidla.

Požadavky na zabezpečení přenosu dat do datové centrály od měřidla lze shrnout takto:

- Data musí být skryta ostatním utilitám v odečtovém procesu. Nejednou se stává, že plynoměr patří jedné společnosti a elektroměr druhé. Někdy je i přenos dat svěřen telekomunikační firmě.
- Data musí být v jednom bloku od každého měřidla, aby je bylo možné přiřadit příslušnému odběrnému místu a k archivaci.
- Data nesmí být pozměněna bez značky, kdo je autorem změny.
- Data musí být v originální formě, ve které je vyslalo měřidlo. Je totiž nutno u případného sporu dokladovat zdrojová data.

Z uvedeného vyplývá, že i zde je nutno použít vhodný model šifrování. Na obr. 3 je obecný princip zabezpečení přenosu dat od měřidla do datové centrály.



Obr. 3: Zabezpečení předávání měřených dat

Na obr. 3 je organizace zabezpečení přenosu dat. Data vzniknou v měřidle pro přenos, jsou zašifrována pomocí certifikátu a odeslána přenosovou cestou do datové centrály. Zde jsou dešifrována a zpracována.

3.3 Ochrana měřidel

Přidělování certifikátů měřidlům je řízeno správcem certifikátů. Správce certifikátů musí vygenerovat ve spolupráci s certifikační autoritou příslušné certifikáty pro měřidla. Správce certifikátů, nebo jím pověřený subjekt, musí mít možnost dopravit příslušné certifikáty k měřidlům a řídit jejich životní cyklus. Stejně musí pečovat o skladiště certifikátů v místě datové centrály. Rozeslání certifikátů je možné standardně přes datovou síť. Pokud by byla nedůvěra k tomuto způsobu (například podezření na prozrazení certifikátu), je možné doručit certifikát přímo například technikem měření.

Navíc je zde síť přímých odečtů do ručních terminálů. I zde je tedy nutno počítat s nějakou formou ochrany dat získávaných z měřidel. Terminály je též možno přenášet certifikáty do měřidel. Ochranu měřidel lze podle úrovně přístupu zabezpečit jako vícečetnou minimálně heslem a hlavně identifikace měřidla skrze certifikát.

3.4 Ochrana koncentrátorů a datových centrál

U koncentrátorů nepředpokládáme vyšší stupeň ochrany než standardní. Tedy ochranu pouze proti pozměňování nastavení zařízení. To může splnit například heslo zpřístupňující pouze přečtení nastavení parametrů koncentrátoru. Změna parametrizace je možná samozřejmě jen přes certifikát.

Zabezpečení datových centrál není nutné řešit v rámci distribuce. Jejich ochrana je řešena v rámci ochrany serverů a datových uložišť interními metodami.

4 Managing bezpečnosti v systémech SMR

V předchozích kapitolách je popsán systém sám, stejně jako jsou naznačeny některé myšlenky řízení bezpečnosti systému. Shrneme-li požadavky na zabezpečení systému SMR, lze popsat požadavky řízení asi takto:

- Systém musí mít, vedle datové centrály, uložiště certifikátů zpravované serverem. Odtud se budou řídit životní cykly jednotlivých certifikátů v systému.

- V příslušné utilitě musí být zřízena certifikační autorita. Pokud se tak nestane je možné použití veřejných certifikačních autorit. Nicméně je nutné

uvažovat s řádem miliónů certifikátů ročně. V tomto případě zřejmě rozhodnou ekonomická hlediska.

- Certifikáty musí být vystaveny pro měřící přístroje s dobou životního cyklu maximálně dva roky. Ty pak musí být doručeny k přístrojům.
- Musí vzniknout certifikáty s krátkým životním cyklem, které budou přidělovány v cechovnách přístrojů a budou sloužit čistě pro montáž měřidel. Pak budou nahrazeny standardním certifikátem s dlouhým životním cyklem.
- Vzhledem k tomu, že montáž měřidel provádí třetí strana, bude nutné vygenerovat certifikáty i pro tyto firmy tak, aby byly schopny zprovoznit měřidlo.
- Bude nutné vytvořit certifikáty pro servisní a kontrolní týmy provádějící záchranné a revizní odečty.
- A další požadavky, které nemusí být řešeny hned po zavedení systému. Třeba statistiky apod.

Do našich úvah musíme zahrnout i komunikační protokoly mezi jednotlivými komponenty. Na obr. 1 jsou označeny písmeny A, B C, D a X. Rozhraní X mezi měřidlem a zákaznickým display zatím není přesně definováno ani na úrovní EU. V tab. **??** jsou souhrnně popsány používané typy přenosových protokolů a komunikačních cest.

Samozřejmě, že většina komunikačních cest má svoji bezpečnostní ochranu. Tuto ochranu budeme chápat jako doplňující. Většinou se jedná o ochranu na linkové a transportní vrstvě OSI.

5 Závěr

Vzhledem implementace systému v následujících letech je nutné vybrat bezpečné protokoly podle jejich vlastností a nasadit je na příslušném místě systému. Zkušenosti s praktickou realizací jsou zatím v začátcích, a tak otázky bezpečnosti teprve přicházejí na pořad dne. Jedna věc je bezpečnost a druhá věc je, jak efektivně tuto bezpečnost řídit a spravovat. To je jistě velká výzva zejména s ohledem na minimalizaci přepočtených nákladů na jedno odběrné místo. Bezpečnostní systémy je nutno budovat nejpozději v první třetině implementace systému v dané oblasti. Samozřejmě, že v té době musí být jasno vše o životních cyklech certifikátů, jejich distribuci atd. Musí být zpracovány scénáře chování při penetraci systému třetí stranou.

Tento článek měl za úkol jen poukázat na jednu specifickou problematiku bezpečnosti systémů a nastínit otázky, které budou muset být vyřešeny v kontextu

celé Evropy. V nasazení systémů SMR jsou nejdále energetici Švédska, případně Anglie. Nicméně jejich hlavním cílem je provozování SMR a na otázky řízení bezpečnosti dochází zatím jen okrajově. Základní zabezpečení je ponecháno na dodavatelích měřidel a ostatní technologie.

Smart metering – nová koncepce měření!

Soňa Netoličková

E-mail: sona.netolickova@cez.cz

Abstrakt

Smart Metering, neboli inteligentní měření je technologie (obvykle pro měření elektrické energie), která rozšíří možnost lidí rozhodovat o svém způsobu využití energií.

Co si vlastně pod pojmem Smart metering představit – je to nástroj pro dálkové ovládání měřidel, nástroj pro průběžné měření spotřeby a dálkový přenos dat o naměřených hodnotách, či údajů o stavech sítě nebo poruchových hlášeních. Cílovým segmentem pro nasazení této technologie jsou běžní zákazníci, tj. malí podnikatelé a obyvatelstvo.

Evropská Unie v červenci 2009 vydala Směrnici Evropského parlamentu a Rady 2009/72/ES o společných pravidlech pro vnitřní trh s elektřinou, ve které se o inteligentních měřicích systémech zmiňuje takto:

Členské státy zajistí zavedení inteligentních měřicích systémů, které podpoří aktivní účast spotřebitelů na trhu s dodávkami elektřiny. Zavedení těchto měřicích systémů může být podmíněno ekonomickým posouzením všech dlouhodobých nákladů a přínosů pro trh a jednotlivého spotřebitele nebo posouzením toho, jaký způsob inteligentního měření je z hospodářského hlediska nejpřiměřenější a nákladově nejefektivnější a jaký harmonogram jejich distribuce je proveditelný. Toto posouzení se provede do 3. září 2012. Pokud národní studie bude vyhodnocena pozitivně, musí být do roku 2020 inteligentními měřicími systémy vybaveno alespoň 80 % spotřebitelů.

Tato koncepce se týká také měření ostatních médií (plynu, tepla, ...), kdy bude možné využít synergií ze společné komunikační cesty – tzv.multiutilitní řešení komunikace.

V posledním desetiletí je největší rozvoj inteligentních měřidel soustředěn především na měření elektřiny – tj. elektroměrů. Do současné doby používáme k měření odebrané elektrické energie převážně mechanické indukční elektroměry, jejichž funkce je založena na Ferrarisově principu. U těchto zařízení je výstup pro zákazníka většinou omezen pouze na měření odebrané energie.

Elektroměr pro Smart metering není jen digitální elektroměr s mikroprocesorem a LCD displejem zobrazující více parametrů. Důležitou součástí inteligentních elektroměrů je komunikační jednotka, která umožní dálkovou správu

měřidla s možností rozšířit nebo vylepšit stávající funkce elektroměru. Dalším prvkem, kterým bývají tyto elektroměry vybavovány v čím dál větší míře, je tzv. elektronicky řízený omezovač výkonu nebo jistič. Diky tomuto prvku je možné dálkově připojit nebo odpojit odběrné místo případně řídit odběr.

Elektroměr pro Smart metering nabízí celou řadu funkcionalit mezi které patří zejména:

- Celková spotřeba energie
- Spotřeba v daném čase
- Připojení/odpojení odběrného místa
- Dálková správa
- Předplacený tarif/tarifní mód
- Informace pro zákazníka
- Změny tarifů
- Řízení odběru
- Odezva v reálném čase
- Ovládání relé zátěže
- Senzory rozpoznání pokusů o ovlivnění měření (odkrytování, přítomnost magnetického pole)
- Zaznamenání a informace o výpadcích

Správa elektroměru tzv. parametrizace předurčuje provoz a některé vlastnosti elektroměru. Při parametrizaci bereme v úvahu výbavu a provedení elektroměru i jeho budoucí nasazení. Při parametrizaci nastavujeme tyto hlavní funkcionality:

- Množství zobrazených údajů na displeji elektroměru a periodu jejich přepínání
- Ovládání relé zátěže – trvale ON/OFF, přepínání dle tarifu
- Nastavení vypínacího výkonu – nastavení limitního výkonu a případného časového zpoždění
- Nastavení sledování výpadků, dovolené poklesy nebo překročení napětí, překročení nastaveného proudu
- Data ukládaná do registrů – možno nastavit více než 1000 registrů
- Ukládané profily, měřené kanály – lze měřit až 8 kanálů (veličin) a v periodě od minut do dnů a ukládat je do paměti
- Parametry pulzního vstupu – z plynoměru, vodoměru, ...
- Přepínání tarifů - kalendářem nebo programem, interním nebo externím HDO
- Editovat nastavení vnitřních kalendářů

Členění architektury technologie

Úroveň 1 – měření, sběr a zpracování dat na odběrném místě, včetně jeho řízení

Tato základní úroveň je tvořena samotnými chytrými elektroměry s různým stupněm jejich technologické výbavy a odlišnými typy komunikačních rozhraní. Tato úroveň je dnes rozšiřována o tzv. Home area network (dále jen HAN), tedy o síť pokrývající nejbližší okolí elektroměru, typicky domácnost, a zařízení v ní spolupracující. Těmi mohou být např. další měřidla (plyn, voda, teplo, atp.), chytré spotřebiče (myčky, ledničky, závlahové systémy) nebo i celé systémy „inteligentní“ domácnosti, které využívají měřidla v této síti jako jedny z mnoha senzorů poskytujících údaje o stavech domácnosti.

Úroveň 2 – sběr dat a řízení infrastruktury

V této architektonické úrovni jsou v aktuálním technologické řešení nejvýznamnějšími prvky takzvané (datové) koncentrátory.

Koncentrátor je malý průmyslový počítač, obvykle velikosti elektroměru, s vlastním operačním systémem a dalším specifickým programovým vybavením, které zajišťuje komunikaci s podřízenými měřidly, automatizaci opakujících se úloh (např. pravidelné odečty) a základní bezpečnost systému. Koncentrátory bývají zpravidla přímo napojeny na datovou centrálu. Víceúrovňové vrstvení koncentrátorů je velmi výjimečné.

Z výše uvedeného je zřejmé, že koncentrátor vlastně zajišťuje funkci prostředníka mezi měřidly a datovou centrálou. Důvodem je nejenom ekonomičnost řešení, kdy jsou využívány nízko-nákladové komunikační technologie poslední míle (RF, PLC) a kompresní algoritmy, ale také distribuce jisté míry inteligence na nižší úroveň systému. Tím je zjištěno nejenom snížení zátěže v datovém centru, ale i zvýšení odolnosti proti výpadkům komunikační infrastruktury.

Úroveň 3 – datová centrála

Jedná se o nejvyšší architektonickou úroveň, která je tvořena informačním systémem složeným z řady různých HW a SW komponent umístěným v datovém centru.

Jádro systému je odpovědné, nejenom za řízení odečtů naměřených dat, jejich validaci (tj. kontrola „odsouhlasení“ odečtených dat – např. porovnání předchozího a nového odečtu), agregaci (úprava změřených „surových“ dat – změna formátů, suma za období např. měsíc), ale i monitoruje (např. sílu signálu), udržuje (např. synchronní čas, kryptografické klíče) a řídí celou podřízenou síť měřidel, koncentrátorů a dalších prvků.

Téměř vždy nestojí takovýto systém osamocen, ale je začleněn do komplexního IT prostředí jen jako jeden z mnoha. Z tohoto důvodu je k základnímu systému vytvořena řada různých rozhraní na okolní systémy, které data ze systému přebírají (např. zákaznické, fakturační) nebo data systému poskytují (např. technické). Současný trend směřující k systémům chytrých distribučních sítí zvyšuje důležitost rozhraní na okolní systémy o další stupeň, a to zejména z důvodu předpokládané zvýšené závislosti na naměřených datech.

Závěrem

Technologická úroveň těchto systémů na území České republiky je průběžně testována od počátku tohoto tisíciletí jednotlivými distribučními společnostmi.

3. září 2012 nedávno bylo. Ministerstvo průmyslu a obchodu vydalo posouzení všech dlouhodobých přínosů a nákladů pro trh a jednotlivé zákazníky při zavedení inteligentních měřicích systémů v elektroenergetice a plynárenství v ČR.

Citace: Z ekonomického posouzení vyplývá, že zavedení inteligentního měření v ČR nelze v současné situaci doporučit. Mnohé z benefitů, které jsou EU očekávány od zavedení AMM, již zákazníci v ČR využívají díky HDO.

Navrhuje se proto:

- zachovat funkčnost současného systému HDO nejméně do roku 2020 a doplnit jej o tarify nepodmíněné blokací spotřebičů ve dvou či třech ekonomických variantách (prahové hodnoty mezi vysokým a nízkým tarifem a jejich časový režim umožňující reakci zákazníků na hladiny cen dodávky elektřiny a zvýšit využívání těchto mechanismů),
- dokončit pilotní projekty AMM a provést jejich vyhodnocení do konce roku 2016,
- na základě vyhodnocení pilotních projektů a v návaznosti na dožívání systému HDO předložit do roku 2018 implementační plán AMM jako součást širšího projektu implementace inteligentních sítí v ČR (v horizontu 2020 až 2030),
- do r. 2017 stanovit národní komunikační standardy, standardy měřicích zařízení a hlavních prvků systému AMM a nastavit technické a legislativní podmínky pro zajištění kybernetické bezpečnosti systému AMM.

Modely a metody výpočtů sítí NN zahrnující fotovoltaické zdroje

Eduard Janeček, Petr Janeček

E-mail: janecek@kky.zcu.cz

1 Úvod

Článek představuje stochastickou metodu výpočtu sítě NN využívající modely odběratelů a také modely fotovoltaických elektráren. Tato metoda výpočtu byla podrobně popsána v [1]. Výhodou tohoto algoritmu je, že kromě očekávaných hodnot je pro každou veličinu (napětí, proud, výkon a ztráty) také počítána směrodatná odchylka a VaR-hodnota, tedy hodnota, kterou veličina překračuje s danou pravděpodobností.

Článek také prezentuje výsledky ověření výpočtů na skutečné testovací síti, na které bylo provedeno porovnání s měřením v síti. Pro účely testování této metody byla vybrána síť s velkým množstvím instalovaných fotovoltaických zdrojů. Pozornost v této síti byla směřována na napětí ve dvou potenciálně kritických místech a sice v rozpojovací skříni R5, kde je očekávané nejnižší napětí, a v místě připojení fotovoltaické elektrárny o instalovaném výkonu 70 kWp, kde je očekáváno nejvyšší napětí v síti. Dalším důležitým ukazatelem provozu je dodaná a odebraná práce do a ze sítě VN. Především přetok výkonu do sítě VN je provozovatelem sítě hodnocen negativně.

Poslední část ukazuje možnosti využití výpočtu pro hodnocení sítí. Metoda umožňuje simulovat chování sítě během roku při jednotlivých meteorologických podmínkách. V praktických úlohách je vhodné zkoumat chovaní soustavy za extrémních podmínek (jasno, zataženo) a za skutečně pozorovaného historického počasí.

2 Popis výpočtů

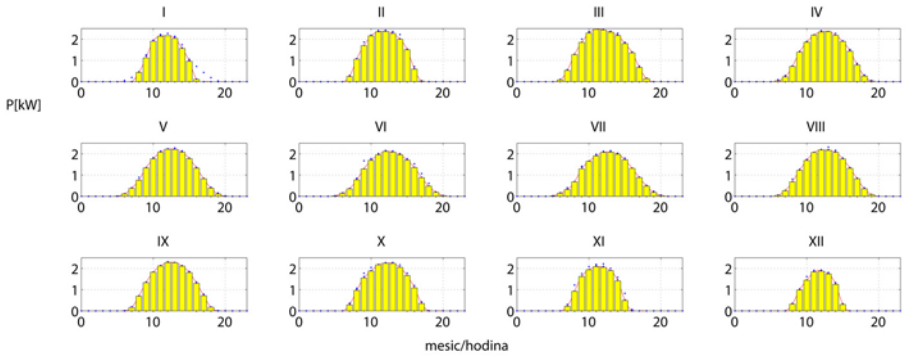
Navržený algoritmus umožňuje výpočet ustáleného stavu sítě. K dispozici jsou očekávané hodnoty a kvantily napětí na všech prvcích sítě, směru a velikosti proudu a přenášeného výkonu ve vedeních a na transformátorech. Výpočet je založen na následujících modelech:

- model sítě,
- model odběratelů,
- model fotovoltaických elektráren.

Model sítě je sestaven na základě exportovaných dat ze systémů provozovatele soustavy. Data obsahují parametry jednotlivých prvků sítě, jako jsou především vedení a transformátory. Tato data také definují konektivitu jednotlivých prvků sítě, místa připojení jednotlivých odběratelů a fotovoltaických elektráren.

Model odběratelů vychází z normalizovaných typových dodávek elektřiny OTE a z koeficientů popisující směrodatnou odchylku odběru. Model je dále parametrizován sazbou a roční spotřebou, které jsou exportovány ze zákaznického systému provozovatele soustavy.

Model fotovoltaické elektrárny vychází z dlouhodobých měření dodávaného výkonu na referenční fotovoltaické elektrárny a je parametrizován velikostí nominálního výkonu elektrárny. Na základě těchto měření byl stanoven normovaný průběh maximálního výkonu během roku. Pro elektrárnu o nominálním výkonu 3kWp jsou tyto průběhy vykresleny na grafu 1.

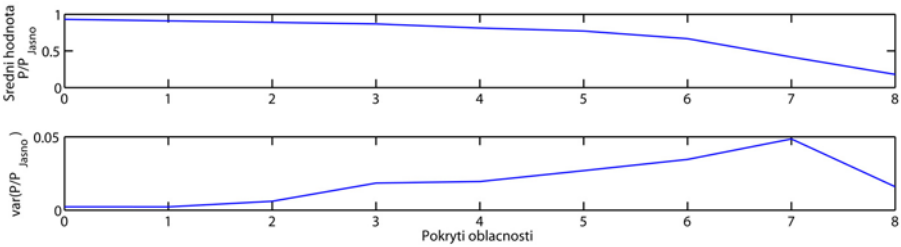


Obr. 1: Maximální výkon FVE o nominálním výkonu 3 kWp

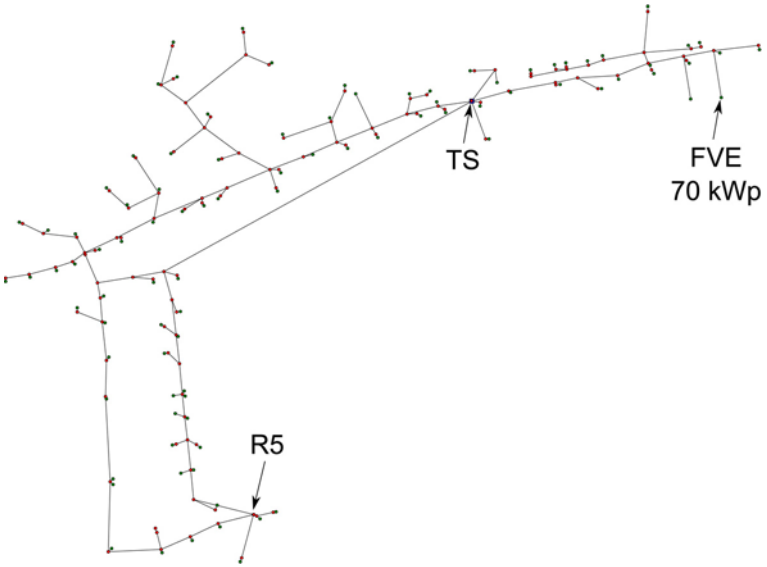
Množství vyrobené energie je dále závislé především na pokrytí oblohy oblačností, což je další vstup modelu. Výpočet je možné provádět buď pro zvolenou hladinu (například ověření provozu v extrémních případech jasno a zataženo) nebo na základě zaznamenaných dat o počasí z meteorologických stanic. Relativní výkon elektrárny pro jednotlivé třídy oblačnosti je zobrazen na obr. 2.

3 Distribuční síť testovací obce

V testovací obci se nachází 99 maloodběratelů, 7 fotovoltaických elektráren o celkovém nominálním výkonu 142 kWp a je připojena přes transformátorovou stanici 250 kVA.



Obr. 2: Normovaný výkon FVE dle pokrytí oblačnosti

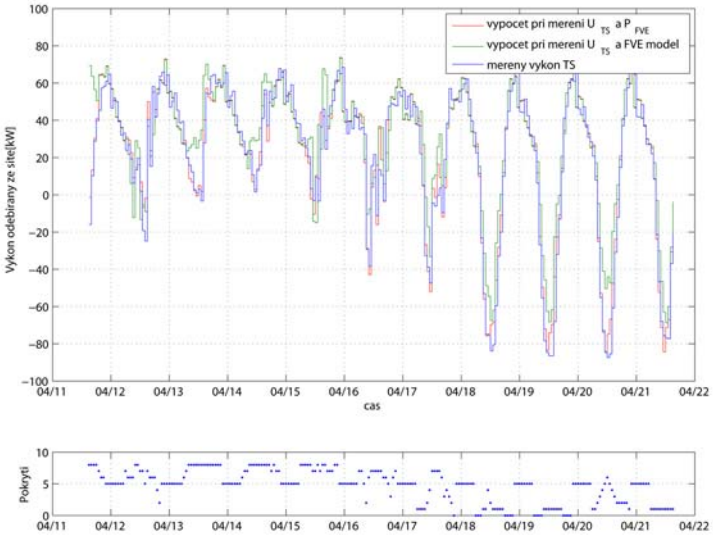


Obr. 3: Testovací obec – schéma vygenerované po načtení dat

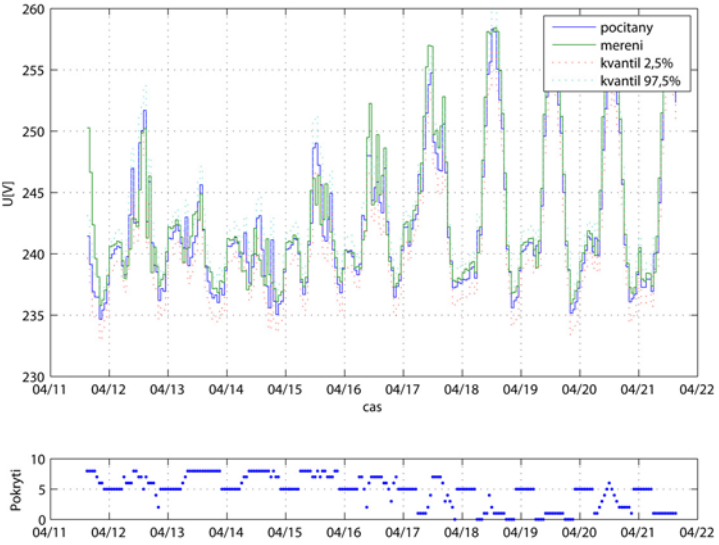
Trafostanice se nachází uprostřed obce, FVE elektrárna v pravé části a R5 na spodním okraji. Pro kontrolu rozvržení a propojení sítě je na obrázku 3 schéma vygenerované na základě dat ze systému GIS.

4 Ověření správnosti výpočtů

V měsících dubnu a květnu 2011 byla provedena v síti testovací obce měření napětí, proudu a výkonu ve významných uzlech sítě. Pro měření bylo zvoleno sekundární vinutí transformátoru, největší FVE s nominálním výkonem 70 kWp a z důvodu předpokladu největšího úbytku napětí rozpojovací skříň R5.



Obr. 4: Výkon na transformátoru ve stanici Obec



Obr. 5: Napětí FVE

Ověření správnosti výsledků výpočtů bylo provedeno pro časový úsek od 11. 4. 2011 do 12. 5. 2011, pro který jsou k dispozici naměřená data i údaje o pokrytí oblačností z nejbližší meteorologické stanice Mikulka (Plzeň-město). V grafech jsou z důvodu přehlednosti zobrazena data jen pro 10 dní. Byly porovnány vypočtené střední hodnoty a jejich kvantily s naměřenými hodnotami pro zvolené veličiny. V grafu na obrázku 4 je zobrazen průběh činného výkonu na transformátoru ve stanici Obec.

Výkon je počítán při měřeném napětí na TR a při měřeném i modelovaném výkonu FVE. Ve spodním grafu je zobrazeno pokrytí oblačností. Jelikož v grafu nejsou významné rozdíly mezi vypočteným a naměřeným výkonem, lze říci, že metoda výpočtu včetně modelu odběratelů a FVE vhodně simuluje výkonové poměry v síti.

Porovnání vypočteného napětí na FVE a R5 při měřeném napětí TR i výkonu FVE je v grafech na obrázcích 5 a 6. Z obou grafů je zřejmé, že algoritmus správně počítá úbytky napětí na vedeních. Průběh napětí na FVE při konstantním napětí TR a měření výkonu FVE je na obrázku 7. Zde není zohledněno kolísání napětí na straně VN a výsledky výpočtů jsou dle očekávání horší.

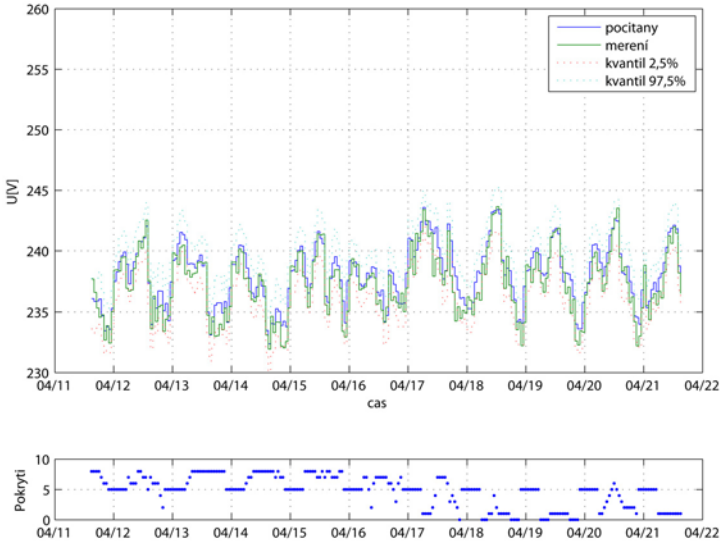
5 Výpočet chodu testovací sítě

Výpočet chodu sítě je proveden pro rok 2011 pro dva extrémní případy počasí: zataženo a jasno po celý rok. Dále pak pro počasí pozorované od dubna do listopadu téhož roku v meteorologické stanici Mikulka. Hlavní pozornost je věnována hodnotě dodávaného i odebíraného výkonu na trafostanici Obec a hodnotám napětí v kritických uzlech sítě (FVE a R5). Pro výpočty se předpokládá konstantní napětí v síti VN. Výkon FVE a spotřeba maloodběratelů jsou modelovány.

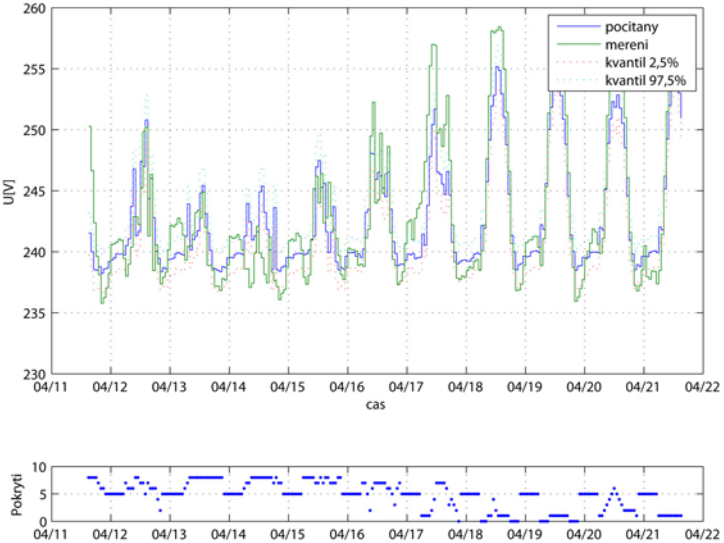
Na obrázku 8 je zobrazen průběh střední hodnoty napětí a příslušných kvantilů pro jasnou a zataženou oblohu. Větší šířka pásma napětí pro jasno je dána rozdílem napětí mezi dnem a nocí, kdy FVE nevyrábí. Průběh napětí na R5 při stejných podmínkách je v grafech na obrázku 9.

Pro oblačnost pozorovanou ve stanici Mikulka je průběh napětí na FVE a R5 znázorněn v grafech 10 a 11. V grafu 10 je výpočet proveden pro aktuální nastavení odbočky trafa (odbočka 0.975) a v grafu 11 je je proveden při stejných podmínkách ale s jinou odbočkou (odbočka 1 a tedy nižší napětí na sekundárním vývodu transformátoru).

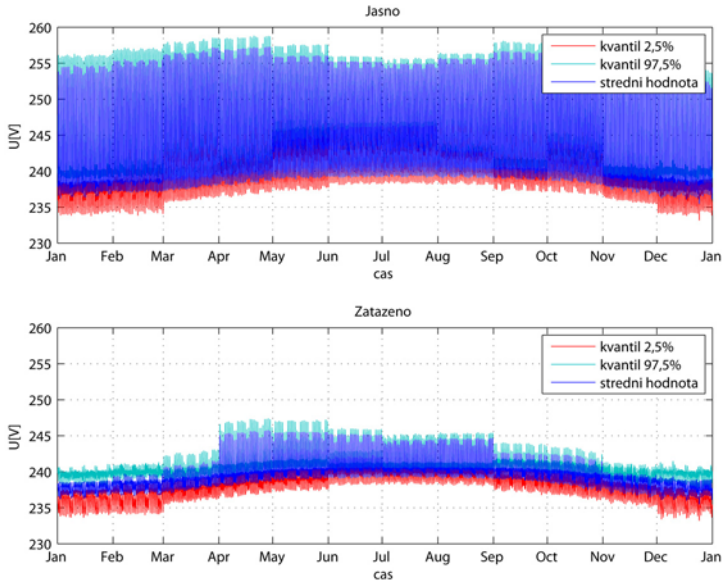
Z výpočtů z granularitou jedna hodina byly vypočteny hodnoty dodané a odebrané el. energie do nebo ze sítě VN pro jednotlivé měsíce. Pro stejná období je vypočteno procento hodinových vzorků nesplňujících normu velikosti napětí v síti. V našem případě překročení napětí 230 V + 10 % na FVE a nedosažení napětí 230 V − 10 % na R5. Hodnoty sledovaných veličin v měsíčních intervalech



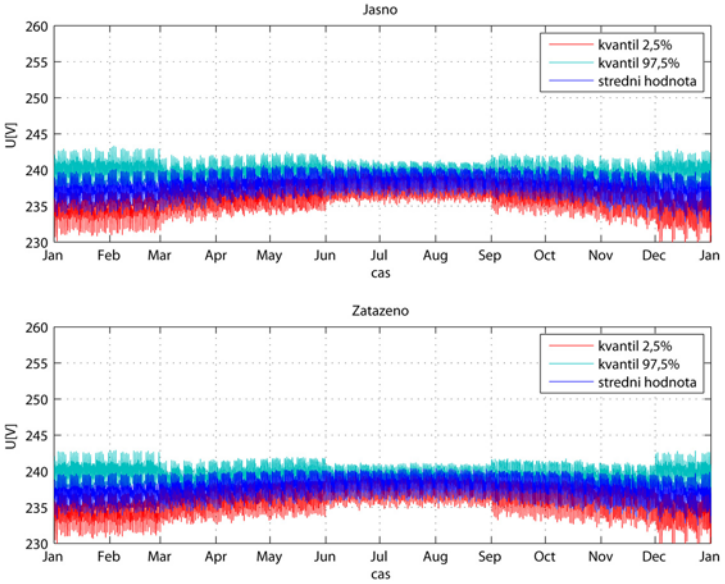
Obr. 6: Napětí R5



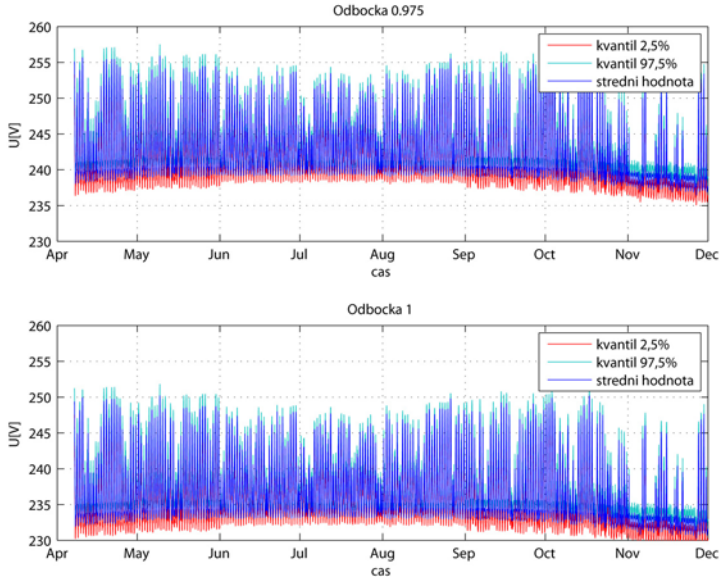
Obr. 7: Napětí FVE, konstantní napětí na TR



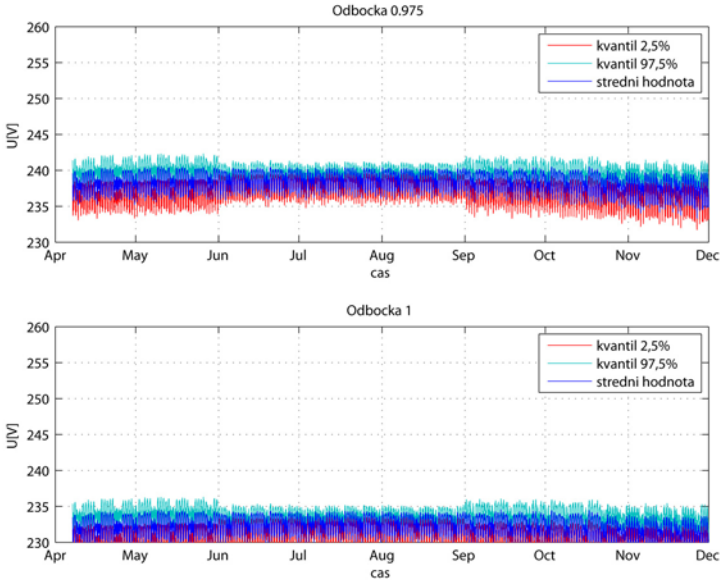
Obr. 8: Napětí na FVE, konstantní pokrytí oblačností



Obr. 9: Napětí na R5, konstantní pokrytí oblačností



Obr. 10: Napětí na FVE, pokrytí Mikulka



Obr. 11: Napětí na R5, pokrytí Mikulka

jsou v tabulkách 1, 2, 3 a 4. Na základě grafů a tabulek, lze konstatovat, že při jasném počasí by bylo významně překračováno nejvyšší povolené napětí na FVE. Při výpočtu se sledovaným počasím je počet případů s přepětím výrazně menší. Naproti tomu je napětí ve stanici R5 vždy v normě. Problém s překročením napětí na FVE by bylo pravděpodobně možné řešit přepnutím odbočky TR.

Tab. 1: Sledované měsíční hodnoty pro jasno

Měsíc	TR výroba [kWh]	TR odběr [kWh]	Přepětí na FVE [%]	Podpětí na R5 [%]
01/2011	−5 923,68	34 468,08	9,65	0,00
02/2011	−7 509,37	26 936,40	16,03	0,00
03/2011	−11 120,73	24 089,73	21,65	0,00
04/2011	−9 701,89	19 646,14	18,72	0,00
05/2011	−10 317,41	16 392,79	17,93	0,00
06/2011	−9 352,96	14 387,08	15,70	0,00
07/2011	−8 773,60	14 452,30	15,42	0,00
08/2011	−9 767,26	15 634,78	16,86	0,00
09/2011	−9 816,38	17 690,50	18,86	0,00
10/2011	−9 450,41	20 697,87	18,63	0,00
11/2011	−6 346,99	30 170,80	8,11	0,00
12/2011	−3 909,46	36 001,66	1,82	0,00

Tab. 2: Sledované měsíční hodnoty pro zataženo

Měsíc	TR výroba [kWh]	TR odběr [kWh]	Přepětí na FVE [%]	Podpětí na R5 [%]
01/2011	0,00	47 487,72	0,00	0,00
02/2011	0,00	40 966,66	0,00	0,00
03/2011	0,00	40 959,90	0,00	0,00
04/2011	0,00	34 175,64	0,00	0,00
05/2011	0,00	30 498,38	0,00	0,00
06/2011	0,00	27 469,15	0,00	0,00
07/2011	0,00	28 614,09	0,00	0,00
08/2011	0,00	29 047,11	0,00	0,00
09/2011	0,00	31 053,04	0,00	0,00
10/2011	0,00	36 936,45	0,00	0,00
11/2011	0,00	41 599,22	0,00	0,00
12/2011	0,00	47 921,36	0,00	0,00

Tab. 3: Sledované měsíční hodnoty pro počasí Mikulka, převod TR = 0,975

Měsíc	TR výroba [kWh]	TR odběr [kWh]	Přepětí na FVE [%]	Podpětí na R5 [%]
04/2011	−2 911,2	18 274,6	5,2	0,0
05/2011	−5 239,5	19 434,8	5,3	0,0
06/2011	−3 517,5	18 051,3	1,9	0,0
07/2011	−2 317,4	19 016,3	0,6	0,0
08/2011	−4 361,4	18 853,1	4,1	0,0
09/2011	−3 834,9	21 258,1	4,2	0,0
10/2011	−3 229,9	27 168,2	3,7	0,0
11/2011	−1 010,2	37 215,1	0,3	0,0
12/2011	−447,6	42 861,5	0,0	0,0

Tab. 4: Sledované měsíční hodnoty pro počasí Mikulka, převod TR = 1

Měsíc	TR výroba [kWh]	TR odběr [kWh]	Přepětí na FVE [%]	Podpětí na R5 [%]
04/2011	−2 893,30	18 312,38	0,00	0,00
05/2011	−5 212,95	19 496,90	0,00	0,00
06/2011	−3 503,28	18 092,29	0,00	0,00
07/2011	−2 304,94	19 055,17	0,00	0,00
08/2011	−4 344,08	18 965,53	0,00	0,00
09/2011	−3 790,45	21 381,22	0,00	0,00
10/2011	−3 033,70	27 674,39	0,00	0,00
11/2011	−756,00	38 141,80	0,00	0,00

6 Závěr

V článku byla představená stochastická metoda výpočtu sítí NN. Pro ověření přesnosti metody byly výsledky výpočtů testovací sítě porovnány s naměřenými hodnotami. Metoda dosáhla přesných výsledků pro výpočet toku výkonu i úbytků napětí. V závěru článku byla ukázána možnost využití výpočtů v praxi. Tato případová studie analyzuje chovaní sítě NN v testovací obci ve třech rozdílných podmínkách a sice za jasného počasí, za zataženého počasí a za počasí pozorovaného během roku 2011 v nejbližší profesionální meteorologické stanici. Z provedených výpočtů je patrné, že při jasném počasí je velmi často překračováno maximální povolené napětí v místě fotovoltaické elektrárny. Je však patrné,

že běžné střídání počasí, které bylo zaznamenáno během roku 2011, tento problém výrazně eliminuje a k překračování povolené meze napětí dochází méně často.

Literatura

[1] Janecek, E., Georgiev, D.: Probabilistic extension of the backward/forward load flow analysis method. *Power Systems, IEEE Transactions on*, 27(2), May 2012, p. 695–704.

PostgreSQL vs. SSD

Tomáš Vondra

E-mail: tv@fuzzy.cz

Abstrakt

SSD disky v poslední době čím dál více pronikají do běžných produkčních konfigurací, a to zejména díky nesporným technickým výhodám, klesající ceně a zvyšující se spolehlivosti. V praxi se však ukazuje že nasazení SSD disků rozhodně nemusí nutně znamenat nárůst výkonu aplikace, případně že zvýšení výkonu neodpovídá vynaloženým prostředkům.

Cílem testů, popisovaných tomto článku a přednášce, bylo srovnat výkon a cenovou efektivitu různých typů a konfigurací úložných zařízení, a to jak z consumer-level tak i enterprise segmentu (podrobněji o použitém hardware viz odstavec 1, včetně základních charakteristik.

K aplikačnímu testování byla využita relační databáze PostgreSQL, na které byly spouštěny testy simulující dva základní typy zátěže. Pro simulaci OLTP zátěže, charakteristické velkým počtem malých transakcí, dotazy přistupujícími k datům přes primární klíče a velkým podílem náhodných I/O operací, byl použit `pgbench`*, nástroj implementující TPC-B-like stress test. Pro simulaci DWH/DSS zátěže, charakteristické malým počtem velkých transakcí, agregačními dotazy nad větším objemem dat a nemalým podílem sekvenčního přístupu k datům, byl použit standardní TPC-H test. Další podrobnosti o testech jsou k dispozici v odstavci 2 na str. 62.*

1 Použitý hardware

V následujícím textu jsou popsány výsledky benchmarků na serveru Dell R810, reprezentujícím poměrně solidní x86 server s velkým počtem jader, SAS disky (15k otáček/minutu) připojenými přes RAID řadič s write cache, a high-end SSD diskem od FusionIO. Reálný objem paměti RAM nehraje až takovou roli, protože její objem byl během benchmarků omezen pomocí parametru jádra „mem“(aby bylo možno použít menší datasety a tedy kratší dobu přípravy a testování). Benchmarky byly prováděny s vhodně naškálovaným datasetem (buď tak aby se pohodlně vešel do paměti, nebo tak aby zabíral cca dvojnásobek RAM). Podrobnější informace o sestavě:

- **CPU**: 4× Intel L7555 @ 1.86 GHz (8 jader každý, tj. celkem 32 fyzických jader)
- **RAM**: 256 GB DDR3 @ 978 MHz (ECC)
- **HDD**: 6× 15 k 300 GB SAS, PERC 6/i RAID s 256 MB write cache
- **SSD**: FusionIO ioDrive Duo (1.28 TB jako 2× 640 GB zařízení)

SAS disky jsou zapojeny do PERC 6/i řadiče (postaveném na řadiči od LSI) a je nad nimi vytvořeno RAID0 pole. FusionIO karta se systému prezentuje jako dvě nezávislá bloková zařízení (`/dev/fioa` a `/dev/fiob`), každé o kapacitě 640 GB (proto „ioDrive Duo“). V rámci benchmarku byl testován jak výkon při využití RAID0 pole postaveného nad oběma zařízeními.

Základní I/O charakteristiky udávající výkon při náhodném a sekvenčním přístupu, zjištěné pomocí nástroje `fio` jsou tyto:

device	rand. read	rand. write	IOPS	seq. read	seq. write
6× 15 k 300 GB SAS RAID	2 MB/s	2 MB/s	390	520 MB/s	480 MB/s
FusionIO (RAID0)	85 MB/s	60 MB/s	21 000	1 400 MB/s	680 MB/s

Během celého benchmarku byl používán pouze souborový systém XFS, vytvoření RAID0 pole nad oběma FusionIO zařízeními tedy může vypadat zhruba takto:

```
# mdadm --create /dev/md0 --level=0 --raid-devices=2 /dev/fioa /dev/fiob
# mkfs.xfs /dev/md0
# mkdir /mnt/fio-raid
# mount /dev/md0 /mnt/fio-raid
```

2 Použité benchmarky

V rámci testování byly využity dva velmi rozdílné benchmarky – `pgbench`, implementující TPC-B-like OLTP stress test nad databází PostgreSQL, a TPC-H benchmark simulující DWH/DSS zátěž. Podívejme se na tyto testy o něco podrobněji.

2.1 pgbench/OLTP benchmark

Pgbench je typickým zástupcem OLTP benchmarků, generujících mnoho „malých“ transakcí a středním (desítky) až velkým počtem klientů (stovky). Standardní TPC-B-like transakce provádějící čtení i zápisy vypadá takto:

```
BEGIN;
  UPDATE pgbench_accounts SET abalance = abalance + :delta WHERE aid = :aid;
  SELECT abalance FROM pgbench_accounts WHERE aid = :aid;
```

```
  UPDATE pgbench_tellers SET tbalance = tbalance + :delta WHERE tid = :tid;
  UPDATE pgbench_branches SET bbalance = bbalance + :delta WHERE bid = :bid;
  INSERT INTO pgbench_history (tid, bid, aid, delta, mtime)
                      VALUES (:tid, :bid, :aid, :delta, CURRENT_TIMESTAMP);
END;
```

a `pgbench` umožňuje pomocí volby „-S“ přepnout do read-only módu, ve kterém je transakce ekvivalentní tomuto jedinému SELECT příkazu:

```
SELECT abalance FROM pgbench_accounts WHERE aid = :aid;
```

Schéma je velmi zjednodušeným modelem banky s účty (`accounts`), pobočkami (`branches`), přepážkami (`tellers`) a historií transakcí (`history`). Podrobnější popis najdete např. na adrese http://www.tpc.org/tpcb/.

Škálování TPC-B benchmark (a tedy i `pgbench`) jako měřítko velikosti datasetu používá počet záznamů v tabulce `accounts`, ke kterému jsou vztaženy i velikosti ostatních tabulek. Velikost je udávána pomocí parametru „scale“ který určuje velikost v násobcích 100 000 (tj. „scale = 1“ znamená 100 000 řádek, „scale = 10“ znamená 1 000 000 řádek apod.), přičemž v prostředí PostgreSQL znamená každých 100 000 řádek cca 15 MB zabraného místa na disku.

Vzhledem k tomu že cílem testování bylo otestovat chování v situaci kdy dataset výrazně převyšuje kapacitu RAM, tak i situaci kdy se dataset do paměti zcela vejde (zejména kvůli read-write zátěži), byly zvoleny následující velikosti:

systém	**RAM**	**scale**	**objem na disku**
R810 (dataset $\ll$ RAM)	96 GB	1 000	15 GB
R810 (dataset $\gg$ RAM)	96 GB	13 000	200 GB

TPS Výsledkem TPC-B benchmarku je hodnota udávající počet transakcí vypořádaných za jednu vteřinu – tj. pokud test poběží 1 minutu a celkem zpracuje 6 000 transakcí, výsledkem benchmarku bude hodnota $tps = 100$ neboť v průměru se zpracovalo 100 transakcí každou vteřinu. Přitom „transakcí“ je míněna celá skupina SQL příkazů – viz příklady uvedené výše. Je důležité si uvědomit že transakce není ekvivalentní právě jedné diskové operaci, neboť zahrnuje načtení několika řádek, zápis update a operace nad transakčním logem, přičemž každá z těchto operací může znamenat jednu či více diskových operací. To kolik a jakých diskových operací je třeba vykonat závisí na velikosti datasetu (zda se vejde do paměti či nikoliv), a zda se jedná o čistě „read-only“ nebo „read-write“ operace.

Ponechme stranou případ kdy se celý dataset vejde do paměti a read-only transakcí – v tom případě (po nacachování celého datasetu) není třeba žádné I/O operace provádět a výkon je tak na zvoleném uložišti nezávislý. Podívejme

se na případy kdy každá transakce znamená alespoň jednu I/O operaci – v tom případě výkon benchmarku výrazně závisí na výkonu pevného disku. Obvykle udávané hodnoty IOPS (počet náhodných operací za vteřinu) jsou tyto:

disk	otáček za min.	teoretická IOPS	obvyklé hodnoty
SATA 7.2 k	7 200	75	75–100
SAS 10 k	10 000	125	125–150
SAS 15 k	15 000	175	175–300

Nenechte se zmást tím že obvyklé hodnoty jsou vyšší než teoretická hodnota IOPS – jedná se mj. o důsledek toho že disky mají propracované optimalizace při zpracování fronty požadavků (NCQ u SATA, TCQ u SAS), díky kterým jsou tyto teoretické hranice schopny překonat. Uváděné hodnoty samozřejmě nezohledňují použití řadiče s write cache (i když dále uvidíme že RAID10 nad 6× 15 k disky může poskytovat cca 6× IOPS jednoho 15 k disku). V tabulce také nejsou uvedeny SSD disky, u kterých se dosahované IOPS velmi liší model od modelu – v zásadě se ale dá říci že se pohybují v tisících pro zápisy a desetitisících pro čtení.

Další informace o výpočtu IOPS najdete např. na en.wikipedia.org/wiki/IOPS.

2.2 TPC-H / DWH/DSS benchmark

TPC-H benchmark je používán k testování DSS/DWH výkonu, tj. velkých dotazů s JOINy, agregacemi apod. a malým počtem klientů (jednotky). Součástí TPC-H benchmarku je 22 vzorových dotazů – nemá cenu zde uvádět jednotlivé dotazy, pro ilustraci se podívejme např. na dotaz č. 13:

```
SELECT
    c_count,
    COUNT(*) AS custdist
FROM
    (   SELECT
            c_custkey,
            COUNT(o_orderkey)
        FROM
            customer LEFT JOIN orders
            ON (c_custkey = o_custkey AND o_comment NOT LIKE '%:1%:2%')
        GROUP BY c_custkey
    ) AS c_orders (c_custkey, c_count)
GROUP BY c_count
ORDER BY custdist DESC, c_count DESC
LIMIT 1;
```

Obr. 1: Ukázka typu dotazu které jsou součástí TPC-H benchmarku

Dotaz pracuje s poměrně velkým množstvím dat (konkrétně s tabulkou objednávek), provádí agregaci, join s jinou tabulkou, řazení apod. tj. operace typické pro datové sklady – DWH (data warehouse) a DSS (decision support system).

Škálování Obdobně jako `pgbench`, i TPC-H benchmark používá škálování testovacího datasetu, který ale není navázán na počet řádek ale spíše na velikost datasetu a určuje velikost vygenerovaných dat v GB. Tj. například „scale = 1“ znamená dataset o velikosti 1 GB, „scale = 100“ dataset o velikosti 100 GB atd. Je důležité mít na paměti že tento objem se týká vygenerovaných hrubých dat (CSV souborů) a nezahrnuje režii databáze (hlavičky stránek a řádek, apod.) ani sekundární struktury jako jsou indexy apod. které mohou velikost databáze klidně zdvojnásobit.

V rámci testování byl objem paměti snížen na 8 GB (pomocí parametru jádra „mem = 8 G“) a byl použit dataset o velikosti 8 GB (tj. se „scale = 8“) který po načtení do DB zabíral včetně indexů cca 16 GB.

typ objektu	velikost
tabulky	10 GB
indexy	5.8 GB

QPH Obdobně jako u TPC-B benchmarku, který udává výsledek jako počet transakcí za vteřinu, i v případě TPC-H benchmarku je výsledek udáván jako počet operací za jednotku času, s tím že operacemi jsou míněny jednotlivé dotazy a jako jednotka času je (vzhledem k délce jejich zpracování) zvolena hodina. Výsledná hodnota tedy znamená „počet dotazů za hodinu“ (queries per hour).

V rámci benchmarků byla testována i situace kdy běží několik dotazů najednou, tak aby bylo vidět jak dobře daný disk (rotační či SSD) škáluje a kdy je jeho I/O kapacita saturována. Benchmark byl tedy spouštěn pro 1, 2, ... až 10 klientů, kteří dotazy spouštěli v náhodném pořadí (permutaci), tak aby se minimalizoval dopad cachování v paměti a simuloval se tak reálný provoz. Proto mají tabulky s výsledky v kapitole 5.2 deset sloupců.

Pro jednotlivé dotazy byl také stanoven limit (konkrétně 10 minut), a jestliže dotaz v tomto časovém limitu nedoběhl (byť jen pro jednoho klienta), byl přerušen a ve výsledcích je uveden jako nedokončený. Toto je třeba brát v potaz i při porovnávání výsledků, ať již při porovnání běhu s různým počtem klientů na stejném zařízení, tak různých zařízení (např. SAS vs. FusionIO) a uvažovat pouze dotazy které doběhly v obou srovnávaných testech.

Jakmile známe kolik dotazů bylo dokončeno, součet průměrných časů pro jednotlivé dotazy a počet paralelně běžících dotazů, můžeme spočítat QPH:

$$QPH = \frac{3\,600}{\textit{sum of average runtimes}} \times \textit{finished queries} \times \textit{parallel clients}$$

Například pokud byl test prováděn s 5 paralelními klienty, z 22 dotazů (spuštěných všemi klienty, každým v náhodném pořadí) bylo vždy dokončeno 20, a součet průměrných časů těchto dotazů je 2 000 sekund, poté

$$QPH = \frac{3\,600}{2\,000} \times 20 \times 5 \doteq 180$$

tj. tato konfigurace dokáže při 5 paralelních klientech zpracovat 180 dotazů (budeme-li uvažovat všech 22 dotazů, bude to samozřejmě méně, v závislosti na tom kdy by dotazy skutečně doběhly pokud bychom je neukončili to 10 minutách).

3 Poměr cena/výkon

Jednou z přirozených otázek kterou si při výběru hardware běžně klademe je také „Dosáhneme s tímto HW potřebného výkonu s minimálními vynaloženými prostředky?“ Pro její zodpovězení ale samozřejmě nestačí informace o hrubém výkonu (ať již „transactions per second“ nebo „queries per hour“), což je pouze polovina potřebné informace - v praxi proto při rozhodování hraje velkou roli výkon v poměru k vynaloženým nákladům, metrika často označovaná jako „poměr výkon/cena“ (dále jen „VC“). Tu je možno zapsat například jako

$$\begin{aligned} VC_{pgbench} &= \frac{transactions\ per\ second}{cena} \\ VC_{tpc-h} &= \frac{queries\ per\ hour}{cena} \end{aligned}$$

a vzhledem k tomu že se pohybujeme v oblasti úložných zařízení, kde je cena standardně vztahována k jednotce objemu (např. 1 GB), tak

$$\begin{aligned} VC_{pgbench} &= \frac{transactions\ per\ second}{cena\ za\ GB} \\ VC_{tpc-h} &= \frac{queries\ per\ hour}{cena\ za\ GB} \end{aligned}$$

přičemž samozřejmě platí že čím vyšší hodnota, tím lépe (za jednotku ceny bylo dosaženo vyššího výkonu). Relevantní VC metriku je pochopitelně třeba volit s ohledem na povahu aplikace, protože zařízení s výbornou (vysokou) hodnotou $VC_{pgbench}$ většinou mají špatnou (nízkou) VC_{tpc-h} a naopak.

Následující tabulka udává základní informace o použitých úložných zařízeních – kapacitu, celkovou cenu a cenu za GB:

zařízení	**cena (USD)**	**kapacita (GB)**	**cena (USD/GB)**
6× 15k SAS 300 GB + PERC 6/i	1 860	900	2.07
FusionIO IoDrive Duo	15 000	1 280	11.72

Ceny uvedené v tabulce vychází z běných ceníkových cen, přičemž při výpočtu ceny SAS diskového pole byly použity tyto ceny: 300 GB 15 k Seagate Cheetah 15.7 stojí 260 USD, řadič PERC 6/i zhruba 300 USD. Při koupi např. v rámci kompletního serveru ale mohou být ceny samozřejmě podstatně nižší.

Ukažme si výpočet skóre na jednoduchém příkladu – předpokládejme že jsme pomocí nástroje `pgbench` naměřili výkon 3 000 tps a spočítejme poměr výkon/cena pro jednotlivé typy uložišť:

disk	**výkon**	/	**cena**
6× 15 k SAS + PERC 6/i	3 000/2.07	$\doteq$	1 449
FusionIO ioDrive Duo	3 000/11.72	$\doteq$	271

Limity modelu Na tomto místě je třeba upozornit že se jedná o poněkud zjednodušený model, který ne zcela odpovídá realitě. Hlavní rozdíly mezi modelem a reálitou jsou tyto:

- Disky (ať již rotační nebo různé formy SSD) se neprodávají s libovolnou kapacitou, ale v jasně daných kapacitách, daných počtem ploten nebo bloků flash paměti. Např. kapacita SATA disků se dnes typicky pohybuje v násobcích 250 GB, u SSD disků jsou to typicky násobky 40 GB (např. u Intel 320). Záleží ale samozřejmě na konkrétním výrobci.

- Dva „stejné“ disky s různou kapacitou nedávají stejný výkon - rotační disky s vyšší kapacitou mají typicky více ploten a/nebo vyšší hustotu záznamu, takže dosahují vyšších sekvenčních rychlostí. Obdobně u SSD disků, kde se skládáním flash bloků zvyšuje sekvenční rychlost. Rychlost náhodného přístupu ale v obou případech většinou zůstává stejná neboť je dána rychlostí otáčení (u rotačních disků) nebo řadičem (u SSD disků).

- V případě jiných způsobů navýšení kapacity (např. vytvořením RAID0 pole nad dvěma disky) může vzrůst výkon v náhodném i sekvenčním I/O, neboť disky mohou seekovat nezávisle. Existují ale i způsoby (např. kombinace kapacit pomocí „LVM linear“ mapování) kdy sekvenční rychlost vzrůstá jen v některých situacích.

- Cena disku většinou nevzrůstá lineárně s kapacitou – např. 1 TB SATA disk stojí dnes přibližně 1.6× tolik co disk 500 GB, nikoliv 2×. To je dáno jednak technickými důvody (elektronika disku stojí pevně danou částku víceméně bez ohledu na kapacitu), ale samozřejmě i trhem kde poptávka po discích se „střední kapacitou“ (dnes právě 1 TB) jsou považovány za standard.

Nepochybně existují i další hlediska ve kterých se uvedený model více či méně liší od reality, a při aplikaci výsledků je třeba brát v potaz také závislost na aktuálních cenách (které vesměs postupně klesají).

4 Důležité faktory

Při aplikaci výsledků prezentovaných v tomto článku na vaši reálnou aplikaci je třeba pečlivě zvážit jak moc chování vaší aplikace odpovídá zde prezentovaným benchmarkům. Podívejme se na několik základních kritérií která je třeba brát v úvahu.

Velikost (aktivní části) datasetu První důležitý aspekt je velikost datasetu, resp. jeho aktivní části. V praxi je totiž běžné že celá databáze je výrazně větší než je kapacita RAM (i několikanásobně), ale drtivou část zabírají historická data ke kterým se přistupuje jen výjimečně, data neaktivních zákazníků apod. V takových situacích ani tak nezávisí na celkové velikosti datasetu, ale právě na velikosti aktivní části. Pokud se tato aktivní část „vejde“ do RAM, bude chování a výkon zcela jiné než v situaci kdy se do RAM nevejde.

U obou benchmarků (TPC-B i TPC-H) byly vždy spouštěny se dvěma velikostmi datasetů – např. pgbench se škálami 1 000 (cca 15 GB) a 13 000 (cca 200 GB), přičemž systém využíval 96 GB RAM, takže první dataset se bez problémů celý vejde do paměti, zatímco druhý nikoliv.

Lepšímu oddělení aktivní a historické části databáze lze napomoci již při designu databázové struktury – např. vhodným partitioningem tabulek tak aby se historická a aktuální data nemíchala (nedocházelo k ukládání do stejných datových bloků).

Charakter přístupu k datům Druhým důležitým hlediskem je zda vaše aplikace k datům přistupuje spíše náhodně či sekvenčně. Velmi zjednodušeně se dá říci že pgbench je benchmark s naprosto náhodným přístupem k datům (v podstatě pracuje s jednotlivými, náhodně vybranými řádky), zatím co TPC-H je silně sekvenčně založen (ale i zde se vyskytuje nezanedbatelné množství náhodných operací, např. v rámci `JOIN` operací prováděných přes `nested loop` s vnořeným `index scan`em apod.).

Zde samozřejmě hraje podstatnou roli velikost aktivní části datasetu, protože vejde-li se dataset do paměti, potom na použitém uložišti v podstatě nezáleží, neboť po nacachování budou všechny požadavky vypořádány z paměti. Toto samozřejmě platí pouze pro read-only přístup k datům – v okamžiku kdy aplikace musí provádět netriviální množství zápisů, je třeba data skutečně fyzicky zapsat na médium, čímž se dostáváme k následujícímu kritériu.

Poměr čtení a zápisů Třetím kritériem je samozřejmě poměr čtení a zápisů – v podstatě se dá říci že čím méně zápisů aplikace dělá, tím lépe může využít potenciál page cache (file system cache v Linuxu) a shared buffers (cache

udržovaná PostgreSQL). Každý zápis znamená nutnost přímé interakce s úložným zařízením (resp. s RAID řadičem ke kterému je připojeno), a to buď v okamžiku `COMMIT`u, nebo později v rámci `CHECKPOINT`. To samozřejmě znamená ve srovnání s RAM značně pomalejší přístup, a to pro všechna současná zařízení (rotační i SSD disky).

Chcete-li maximálně využívat možnosti cachování, snažte se na aplikační úrovni minimalizovat počet zápisů – není výjimkou že inteligentními úpravami (např. neprovedením `UPDATE` pokud se ve formuláři nezměnila žádná data) lze zněkolikanásobit výkon aplikace.

Další kritéria Výše uvedená kritéria považuji za nejdůležitější, nejsou ale samozřejmě jediná – roli hraje mnoho dalších faktorů. Např. počet spojení a jak rychle počet spojení dokáže saturovat I/O zařízení, nebo to jakou část času aplikace reálně tráví v databázi (což je důležité pro výpočet „business transaction“ a odhad reálných uživatelů které je aplikace schopna podporovat). A tak dále.

5 Výsledky testů

5.1 TPC-B

Jak již bylo uvedeno v kapitole 2.1, OLTP výkon byl testován pomocí pgbench se dvěma různými velikostmi datasetů – 1 000 (cca 15 GB, který se bez problémů vejde do RAM) a 13 000 (cca 200 GB, zhruba 2× kapacita RAM), a vždy byl testován read-only i read-write výkon. Následující výsledky jsou ve všech případech získány s 32 klienty (odpovídá počtu fyzických jader CPU), tj. pomocí těchto příkazů (read-only a read-write):

```
$ pgbench -S -c 32 -j 16 -S -T 900 testdb
$ pgbench -c 32 -j 16 -T 900 testdb
```

Kromě těchto dvou základních parametrů (škála, read-only/read-write) byl během testu zkoumán vliv umístění následujících tří základních částí databáze na různá média, s cílem prozkoumat které médium nejlépe odpovídá způsobu přístupu k jendotlivým částem (např. zápis/čtení, náhodný/sekvenční apod.). Konkrétně se jedná o tyto části:

- **datové soubory** – soubory obsahující vlastní data tabulek (včetně pomocných struktur jako jsou např. visibility map a free space map)
- **indexy** – data indexů, vytvořených nad tabulkami
- **transakční log** – známý také jako transakční log, od předchozích dvou se výrazně liší charakterem přístupu (pouze sekvenční zápisy, ale s velkými nároky na latenci `fsync` operace)

Následující grafy tak vždy pro každou velikost datasetu srovnávají 8 variant rozdělení těchto částí na jednotlivá datová média (diskové pole 6× 15 k SAS a FusionIO). Podívejme se nejdříve na menší (15 GB) dataset, jehož výsledky jsou znázorněny na obrázku 2.

Read-only výkon je velmi vyrovnaný, neboť po úvodním nacachování dat během „warmupu“ jsou již transakce vypořádávány pouze z RAM. Všechny výsledky se pohybují nad 145 000 transakcemi za vteřinu Rozdíly (byť někdy ne zcela zanedbatelné), jsou spíše náhodný šum který se na read-only nacachovaném datasetu projevuje poměrně výrazně. Podívejme se však pouze na read-write výkon – viz obrázek 3.

Zde je situace zcela jiná – rozdíly mezi jednotlivými variantami jsou poměrně značné. Z grafu je zřejmé že víceméně nezáleží na umístění datových souborů ani indexů, i když samozřejmě FusionIO dává cca o 10 % vyšší výkon než SAS disky. Primární parametr na kterém záleží je umístění transakčního logu vzhledem k datovým souborům – v situaci kdy jsou obě tyto části umístěny na stejném zařízení, je výkon výrazně horší než v případech kdy jsou umístěny na samostatných zařízeních. Přitom je celkem jedno zda jsou datové soubory na FusionIO a WAL na SAS discích či naopak – záleží jen na tom aby se `fsync` operace na transakčním logu, které jsou velmi citlivé na latenci, a zápisy iniciované pravidelnými checkpointy. Později uvidíme že v případě datasetů větších než kapacita RAM na tomto již velmi záleží.

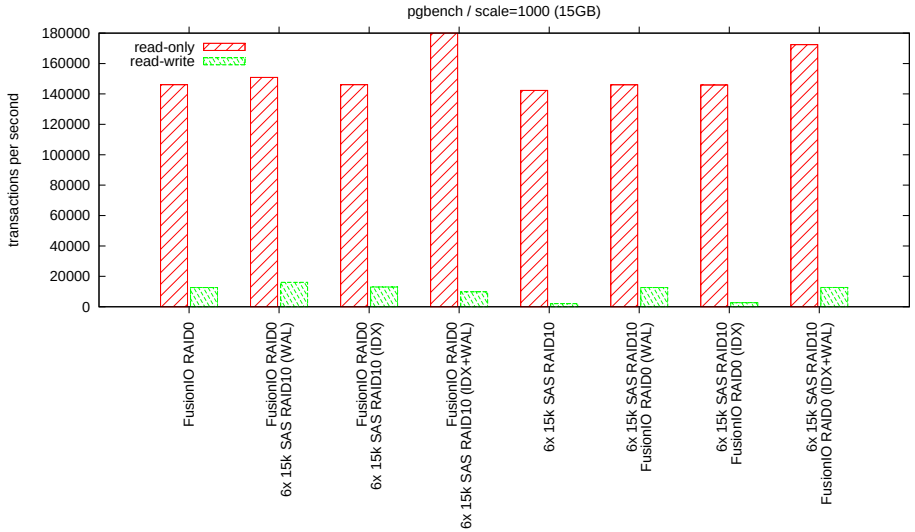
Podívejme se nyní na metriku výkon/cena, jejíž výpočet je ale vzhledem k rozdělení dat na několik médií současně, poněkud komplikovanější. Výpočet ceny/GB pro jednotlivé kombinace je jednoduchým váženým součtem, přičemž se počítá se 60 % kapacity pro datové soubory, 25 % na indexy a 15 % na WAL, což ve výsledku dává ceny uvedené v tabulce 1.

Pokud se toto zkombinuje s read-write výsledky na 15 GB datasetu, získáme graf (4).

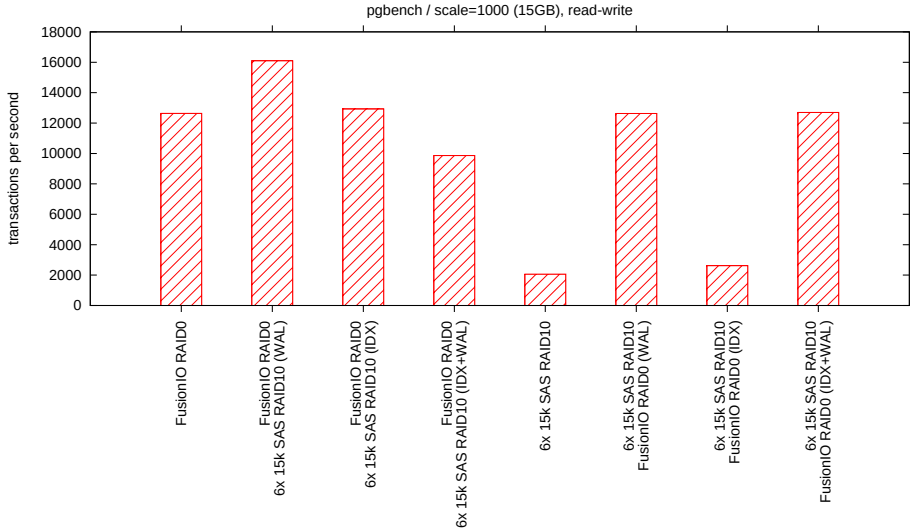
Z něj je zřejmé že (stačí-li vám podávaný výkon), je ekonomicky nejvýhodnější tradiční diskové SAS s transakčním logem odděleným na FusionIO. V zásadě stejný výkon (a lepší výkon/cena metrika) lze očekávat v případě umístění na další SAS diskové pole se samostatným řadičem, write cache apod.

Tab. 1: Tabulka s cenami za GB pro různé kombinace disků

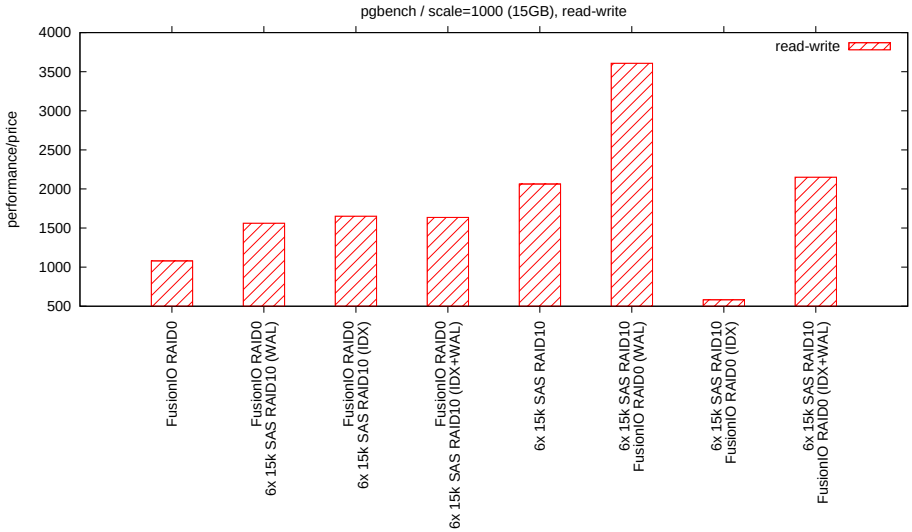
kombinace	**15 k SAS**	**FusionIO**	**cena/GB**
FusionIO RAID0	0 %	100 %	11.7
FusionIO RAID0 / WAL on 15 k SAS	15 %	85 %	10.3
FusionIO RAID0 / IDX on 15 k SAS	25 %	75 %	9.3
FusionIO RAID0 / IDX+WAL on 15 k SAS	40 %	60 %	7.9
15 k SAS RAID10	100 %	0 %	2.0
15 k SAS RAID10 / WAL on FusionIO	85 %	15 %	3.5
15 k SAS RAID10 / IDX on FusionIO	75 %	25 %	4.5
15 k SAS RAID10 / IDX + WAL on 15 k SAS	60 %	40 %	5.9



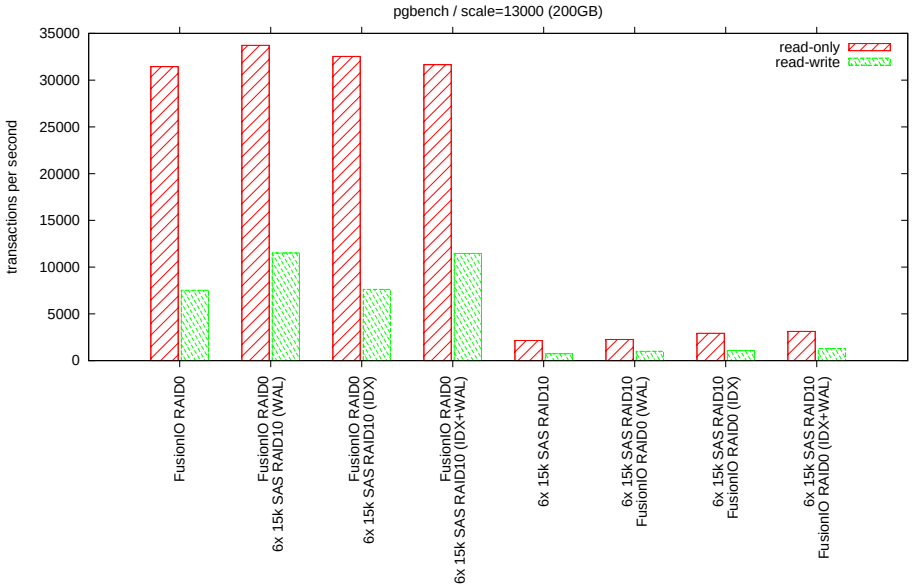
Obr. 2: Výsledky pgbench na 15 GB datasetu (scale = 1 000)



Obr. 3: Výsledky pgbench v read-write módu na 15 GB datasetu (scale = 1 000)



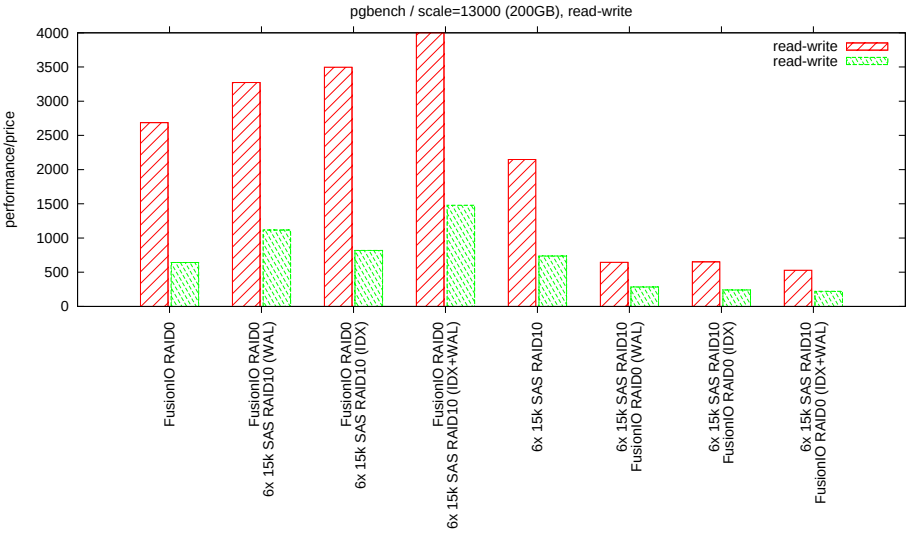
Obr. 4: Metrika výkon/cena pro read-write pgbench na 15 GB datasetu



Obr. 5: Výsledky pgbench na 200 GB datasetu (scale = 13 000)

Podíváme-li se na výsledky benchmarků nad větším datasetem, uvidíme zhruba to co je na obrázku 5.

Na první pohled je zřejmé že situace je velmi odlišná od menšího datasetu, a výkon se zásadně liší dle toho kde jsou umístěny datové soubory, a FusionIO dává daleko lepší výkon. U read-write zátěže opět závisí na oddělení transakčního logu na samostatný device. Pokud do těchto výsledků promítneme cenu, stejně jako u menšího datasetu, získáme graf 6.



Obr. 6: Metrika výkon/cena pro pgbench na 200 GB datasetu

Je zřejmé že pro OLTP dataset převyšující dostupnou paměť je volba FusionIO velmi efektivní – výkonnostně i finančně, samozřejmě jen pokud podávaný výkon skutečně potřebujete.

Jen jedna poznámka ke grafu 6 – výhodnost přesunu indexů a WAL je jen a pouze důsledkem snížení ceny za GB a skutečnosti že indexy jsou udržovány v paměti. V případě že by tlak na paměť byl vyšší a indexy by nebyly udržovány v paměti, docházelo by ke kolizím se zápisy do WAL logu a mohlo by se to negativně podepsat na read-write výkonu.

5.2 TPC-H

Nejdříve se podívejme na kompletní výsledky benchmarku na rotačních discích, či přesněji na RAID10 nad 6× 15 k SAS disky s řadičem, uvedené v kompletní podobě v tabulce 3, a v redukované podobě (jen dokončené dotazy) v tabulce 4.

Tab. 2: Výsledky pgbench na 15 GB a 200 GB datasetu

konfigurace	**scale = 1 000**		**scale = 13 000**	
	read-only	read-write	read-only	read-write
FusionIO RAID0	146 033	12 635	31 444	7 526
FusionIO RAID0/ 6× 15 k SAS RAID10 (WAL)	150 870	16 099	33 720	11 495
FusionIO RAID0/ 6× 15 k SAS RAID10 (IDX)	146 044	12 932	32 535	7 623
FusionIO RAID0/ 6× 15 k SAS RAID10 (IDX +WAL)	180 000	9 861	31 651	11 463
6× 15 k SAS RAID10	142 361	2 062	2 148	736
6× 15 k SAS RAID10/ FusionIO RAID0 (WAL)	145 988	12 629	2 260	996
6× 15 k SAS RAID10/ FusionIO RAID0 (IDX)	145 876	2 622	2 930	1 081
6× 15 k SAS RAID10/ FusionIO RAID0 (IDX+WAL)	172 413	12 698	3 122	1 308

Tab. 3: kompletní výsledky TPC-H benchmarku na rotačních SAS discích

	clients									
query	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	399	411	425	433	443	460	497	501	526	–
2	16	21	39	48	50	64	72	79	97	105
3	50	58	75	111	157	236	222	264	369	325
4	72	114	164	215	210	280	299	292	337	–
5	52	147	107	209	258	322	358	313	410	–
6	46	158	225	257	285	323	316	312	386	393
7	117	170	214	247	285	356	436	-	451	455
8	63	115	194	195	252	286	339	360	476	427
9	–	–	–	–	–	–	–	–	–	–
10	72	118	142	245	312	341	348	398	458	450
11	9	16	23	28	44	54	53	77	63	85
12	43	70	130	140	166	190	228	258	250	299
13	43	50	57	63	76	67	83	94	94	98
14	35	77	187	195	261	287	351	–	299	350
15	91	409	258	–	–	–	–	–	–	–
16	52	53	56	55	60	60	63	63	68	69
17	359	365	449	505	510	–	–	–	–	–
18	120	172	267	278	379	441	–	–	–	–
19	37	73	98	121	138	176	153	158	180	267
20	127	180	189	214	336	311	382	335	–	–
21	–	–	–	–	–	–	–	–	–	–
22	6	6	6	11	11	16	16	14	16	17
# finished	20	20	20	19	19	18	17	15	16	13
runtime	1 809	2 781	3 304	3 570	4 233	4 271	4 217	3 520	4 482	3 338
QPH	40	52	65	77	81	91	102	123	116	140

Tab. 4: redukované výsledky (bez nedokončených dotazů) TPC-H benchmarku na rotačních SAS discích

	clients									
query	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
2	16	21	39	48	50	64	72	79	97	105
3	50	58	75	111	157	236	222	264	369	325
6	46	158	225	257	285	323	316	312	386	393
8	63	115	194	195	252	286	339	360	476	427
10	72	118	142	245	312	341	348	398	458	450
11	9	16	23	28	44	54	53	77	63	85
12	43	70	130	140	166	190	228	258	250	299
13	43	50	57	63	76	67	83	94	94	98
16	52	53	56	55	60	60	63	63	68	69
19	37	73	98	121	138	176	153	158	180	267
22	6	6	6	11	11	16	16	14	16	17
# finished	11	11	11	11	11	11	11	11	11	11
runtime	437	738	1 045	1 274	1 551	1 813	1 893	2 077	2 457	2 535
QPH	91	107	114	124	128	131	146	153	145	156

Tab. 5: kompletní výsledky TPC-H benchmarku na FusionIO ioDrive Duo (SSD)

	clients									
query	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	398	399	401	420	422	434	353	448	447	466
2	14	15	14	14	15	16	13	17	18	18
3	56	53	55	59	63	64	50	69	71	75
4	32	40	38	43	46	49	36	59	67	79
5	44	62	68	61	64	76	54	80	83	76
6	60	48	59	77	65	80	56	92	80	105
7	153	128	123	131	144	132	117	128	137	153
8	98	77	65	64	77	76	66	89	90	86
9	158	263	343	373	402	–	–	–	–	–
10	94	73	77	89	85	93	75	102	104	119
11	8	8	9	8	10	10	9	12	12	12
12	43	44	45	56	48	54	43	55	57	57
13	43	44	45	46	48	50	41	55	52	53
14	64	42	47	51	53	59	47	73	73	72
15	208	149	125	157	157	141	136	170	179	246
16	52	52	53	54	55	54	48	55	58	58
17	201	213	213	217	230	247	191	261	275	285
18	127	130	128	150	141	141	119	155	185	206
19	41	40	45	53	49	47	42	53	57	59
20	171	131	153	131	125	125	122	153	144	147
21	310	324	323	377	400	436	–	–	–	–
22	4	5	5	5	5	5	7	6	6	6
# finished	22	22	22	22	22	21	20	20	20	20
runtime	2 379	2 338	2 434	2 634	2 703	2 389	1 627	2 132	2 196	2 378
QPH	33	68	98	120	147	190	310	270	295	303

Tab. 6: redukované výsledky (bez nedokončených dotazů) TPC-H benchmarku na FusionIO ioDrive Duo (SSD)

	clients									
query	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
	1	2	3	4	5	6	7	8	9	10
1	398	399	401	420	422	434	354	448	447	466
2	14	15	14	14	15	16	13	17	18	18
3	56	53	55	59	63	64	50	69	71	75
4	32	40	38	43	46	49	36	59	67	79
5	44	62	68	61	64	76	54	80	83	76
6	60	48	59	77	65	80	56	92	80	105
7	153	128	123	131	144	132	117	128	137	153
8	98	77	65	64	77	76	66	89	90	86
10	94	73	77	89	85	93	74	102	104	119
11	8	8	9	8	10	10	9	12	12	12
12	43	44	45	56	48	54	43	55	57	57
13	43	44	45	46	48	50	41	55	52	53
14	64	42	47	51	53	59	47	73	73	72
15	208	149	125	157	157	141	136	170	179	246
16	52	52	53	54	55	54	48	55	58	58
17	201	213	213	217	230	247	191	261	275	285
18	127	130	128	150	141	141	119	155	185	206
19	41	40	45	53	49	47	42	53	57	59
20	171	131	153	131	125	125	122	153	144	147
22	4	5	5	5	5	5	7	6	6	6
# finished	20	20	20	20	20	20	20	20	20	20
runtime	1 911	1 753	1 768	1 886	1 902	1 953	1 628	2 132	2 195	2 378
QPH	38	82	122	153	189	221	310	270	295	303

Za pozornost stojí zejména poměrně špatné škálování, neboť 10 klientů má ve srovnání s 1 klientem méně než dvojnásobnou hodnotu QPH (a kdybychom započetli i nedokončené dotazy, bylo by to ještě horší). To je důsledkem poměrně rychlé saturace náhodnými I/O operacemi, kterých rotační disky mnoho nezvládnou.

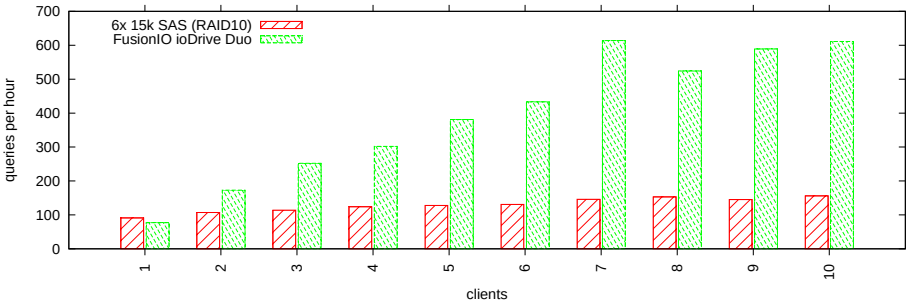
Nyní se podívejme na výkon s FusionIO SSD diskem, opět nejdříve kompletně včetně nedokončených dotazů, které jsou uvedeny v kompletní podobě uvedeny v tabulce 5, a v tabulce 6 sou opět uvedeny jen dokončené dotazy.

Již na první pohled je zřejmé že FusionIO karta škáluje daleko lépe, protože 10 klientů dává ve srovnání s jedním klientem cca 8× výkon.

K tomu abychom však mohli výsledky pro různá zařízení srovnat lépe. musíme uvažovat pouze dotazy dokončené na obou – SAS discích i SSD. Nemá cenu zde znovu uvádět tabulky s kompletními detailními výsledky, ale uvedeme až řádky s agregovanými výsledky – viz tabulka 7 a pro ilustraci také graf na obrázku 7.

Tab. 7: srovnání výsledků TPC-H benchmarku na SAS i FusionIO discích

	clients									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
runtime / SAS	437	738	1 045	1 274	1 551	1 813	1 893	2 077	2 457	2 535
runtime / FIO	513	459	472	525	520	549	451	605	605	648
QPH / SAS	91	107	114	124	128	131	146	153	145	156
QPH / FIO	77	173	252	302	381	433	614	524	589	611



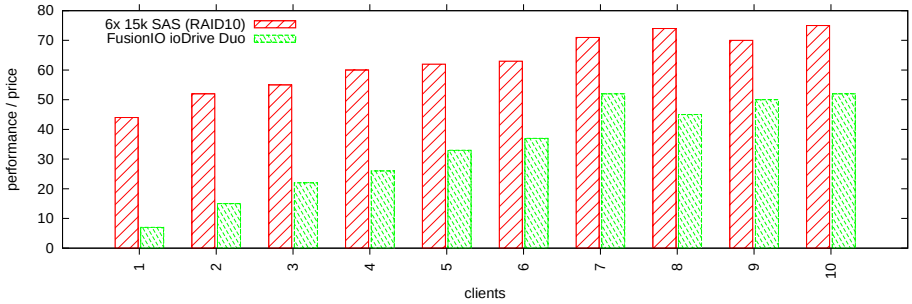
Obr. 7: srovnání výsledků TPC-H benchmarku na SAS i FusionIO discích

Z tabulky a grafu je poměrně pěkně vidět že diskové pole je saturováno poměrně záhy a s růstem klientů roste jen minimálně, zatímco SSD disk je schopen vyřizovat pořadavky i od většího počtu klientů. Jasně se tak projevuje schopnost SSD disků zvládat mix náhodných a sekvenčních operací, zatímco na SAS disky mají náhodné požadavky poměrně značný negativní vliv.

Promítneme-li dále do těchto výsledků cenu, tj. spočteme-li poměr výkon/cena tak jak je uvedeno v odstavci 3, získáme tabulku 8 a graf 8, ze kterých je názorně vidět že náklady na výkon u FusionIO disku jsou znatelně vyšší, a z tohoto pohledu se tedy dá říci že SAS disky jsou pro tuto zátěž cenově efektivnější. Budeme-li u RAID10 diskového pole předpokládat lineární nárůst výkonu v závislosti na počtu disků, potřebovali bychom cca 4× větší diskové pole (24 disků) aby se vyrovnalo jedné FusionIO kartě, a tedy 4× vyšší celkovou cenu (ale při zachování ceny za GB). To sebou samozřejmě nese další komplikace, protože i kdybychom pořídili disky s menší kapacitou (a tedy nižší cenou), stále se jedná o 24 disků které se již nevejdou do běžného rackového serveru a musí být v samostatném boxu, atd.

Tab. 8: srovnání metriky výkon/cena TPC-H benchmarku na SAS i FusionIO

	clients									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
VC / SAS	44	52	55	60	62	63	71	74	70	75
VC / FIO	7	15	22	26	33	37	52	45	50	52



Obr. 8: srovnání metriky výkon/cena TPC-H benchmarku na SAS a FusionIO

6 Závěr

Prvním předpokladem pro zvýšení výkonu po změně diskového subsystému je že úzké hrdlo aplikace skutečně leží v I/O vrstvě – např. pokud je vaše aplikace omezena CPU, bude změna subsystému zbytečná nebo dokonce kontraproduktivní. Druhým, neméně důležitým, předpokladem pro správný výběr typu disků je znalost aplikace, zejména charakteru jakým využívá I/O. Výsledky předchozích testů lze shrnout do několika následujících (velmi zjednodušených) pravidel.

V případě **převažujícího sekvenčního přístupu** (v podstatě nezáleží zda read či write) se jako velmi efektivní jeví rotační disky, obvykle ve vhodné RAID konfiguraci a případně s RAID řadičem s write cache. Vhodně zvolená varianta d iskového pole umožňuje dosahovat velmi vysokého sekvenčního výkonu, a v případě využití write cache dokáže zápisy absorbovat s minimální latencí (což je výhodné např. pro zápisy do transakčního logu). S narůstajícím objemem náhodných I/O operací ale výkon polí založených na rotačních discích výrazně klesá.

S narůstajícím podílem **náhodného přístupu** získávají převahu SSD disky, a to jak v hrubém výkonu tak i poměru cena/výkon. V případě že aplikace data převážně čte, je možno zvážit rozšíření systému o další RAM (tak aby se data četla z diskové cache a nikoliv z disků), ale to mnohdy není možné – ať již kvůli objemu datasetu nebo kvůli převažujícím zápisům. V podstatě platí že čím je objem aktivní části datasetu větší, tím jsou SSD disky výhodnější.

Jak ale ukazuje např. odstavec 5.1, velmi efektivní může být vhodná kombinace obou variant, dle typu I/O potřebného pro danou část aplikace. V daném případě rotačních disků s RAID řadičem pro WAL (transakční log), vyžadující sekvenční zápisy s minimální latencí, a SSD disků pro datové sobory, vyžadující náhodný přístup.

Než ale začnete uvažovat o radikální změně diskového subsystému s cílem zvýšit výkon, zkuste se zamyslet nad konfigurací (např. checkpointingu v případě PostgreSQL) a případně i nad implementací problematických částí aplikace. Zkušenosti ukazují že toto je mnohdy nejjednodušší a nejlevnější krok.

Sloupcové databáze

Pavel Stěhule

E-mail: pavel.stehule@gmail.com

V posledních několika letech jsme si mohli všimnout, že kromě dominantních klasických relačních databází (Oracle, DB2, MSSQL, MySQL, PostgreSQL, Firebird) se můžeme čím dál častěji setkávat i s jinými typy databází. Zatím to jsou spíše vlašťovky, nicméně dá se čekat, že se s nimi budeme setkávat čím dále častěji a v některých oborech je používání těchto nových databází denním chlebem vývojářů (NoSQL databáze pro www aplikace). Také to bude zhruba 15 let, co se primárně na tyto databáze zaměřuje výzkum – a komerční produkty založené na tomto výzkumu mají za sebou prvních 5 let nasazení v ostrém provozu (koncem devadesátých let to byly proudové databáze, po roce 2000 pak sloupcové databáze, nyní se výzkum zaměřuje na databáze polí a masivně distribuované databáze).

Proč se tyto databáze neobjevily před deseti, dvaceti roky, ale až v posledních pěti letech? V posledních pěti letech došlo k neuvěřitelnému nárůstu sociálních sítí, vzrostl význam aplikací, které bez internetu nemohou existovat – eshopy, systémy podpory. Zároveň, díky cloudům, je možné relativně levně získat ohromný výpočetní výkon. Také došlo k poklesu cen pamětí a záznamových médií. Globální sociální sítě, globální aplikace dokáží generovat terabyty dat denně a soudobí uživatelé zdaleka nejsou tak trpěliví, jako uživatelé před deseti, dvaceti nebo třiceti lety. Databáze, které byly navrženy koncem sedmdesátých let a optimalizovány pro vnitropodnikové systémy a pro hw předchozích třiceti let nezvládají zátěž, kterou dokážou vyprodukovat globální aplikace a navíc nedokáží využít moderní cenově dostupnou infrastrukturu cloudů. Laickou reakcí na problémy relačních databází v prostředí www aplikací byla okamžitá popularita NoSQL databází.

Klasické databáze vychází z konceptu univerzálního databázového serveru, který dokáže efektivně pokrýt velkou množinu úloh. Databáze musela být univerzální, protože nebylo ekonomické (ekonomicky únosné) mít více databázových serverů. Zrovna tak nebylo distribuce dat napříč databázovými servery nebyla jednoduchá – a vzhledem k rychlosti sítí mohla představovat úzké hrdlo. Atypickou relační databází byla MySQL, která v prvních verzích byla optimalizována se zřetelem na rychlost bez ohledu na spolehlivost. Pokud dojde k snížení ceny za hw (lze nakoupit více serverů), za média (nemusíme řešit duplicitní uložená

data) a z rychlení přenosu dat po síti (zrychlení distribuce dat), pak se docela logicky musí na scéně objevit jednoúčelové databáze. Tyto databáze řeší většinou pouze jeden typ úloh, ale vzhledem k tomu, že pro tento typ úloh se volí formát uložení dat, tak úlohy, pro které jsou optimalizované, dokáží provést velice efektivně a rychle.

Toto nové paradigma akceptuje i prof. Michael Stonebraker ve svém článku „The End of an Architectural Era (It's Time for a Complete Rewrite)", kde představuje nový koncept jednoúčelových databází a v rámci tohoto konceptu databázi H-Store. Databáze, které architektonicky vychází z konceptu projektu R fy IBM narazily na své limity a je na čase přijít s novými koncepty. V tomto článku z roku 2007 také reviduje své názory, které propagoval předchozích 20 let – akceptuje skutečnost, že relační model nemusí být pro všechny úlohy, a zrovna tak, že SQL není nutné mít všude (podobný Stonebraker článek „One Size Fits All: An Idea whose Time has Come and Gone" vyšel ještě o dva roky dříve). Mnoho aplikačních knihoven dnes efektivně odstiňuje vývojáře od SQL vrstvy – SQL slouží pouze jako komunikační protokol, který je ovšem pak zbytečně rozsáhlý a navíc je, díky svému návrhu orientovanému na čitelnost, příliš náročný na strojové zpracování. Pro řadu úloh postačí pro přístup k datům jednodušší interface – např. model, který známe z Ruby on Rails. Tento článek odpovídal akcentu, který se zhruba před 5 roky kladl na NoSQL databáze (ačkoliv vycházel z výzkumu, který Stonebraker realizoval na MIT po roce 2000). Jednoduché NoSQL databáze vznikly v reakci na těžké problémy s výkonem a provozem klasických databází v prostředí velkých aplikací sociálních sítí, přičemž ale tyto databáze byly bezpochyby používány velmi netradičně (a z pohledu klasika nevhodně). Dynamika sociálních sítí vedla k velice pružnému (a jednoduchému) schématu uložení dat v dvojicích (atribut, hodnota) a to je pro klasické relační databáze navrhované pro práci s entitami a složitějšími relacemi nevhodné. Po roce 2005 vznikla řada NoSQL databází, které měly společné několik vlastností – primárně nerelační model (key/value) a koncept map/reduce - MongoDB, CouchDB. Tyto databáze mají blíže ke konceptu objektových databází, které Stonebraker dříve tvrdě kritizoval „Third-generation Database System Manifesto" (1990). Přestože se Stonebraker vzdal některých svých názorů (na relační model a SQL), stále zůstává ortodoxní ve svém důrazu na ACID (kterého se některé NoSQL databáze s cílem vyššího výkonu vzdaly). V roce 2011 publikoval několik článků a rozhovorů, kde kritizoval návrh NoSQL databází – „Map/Reduce is a giant step backward in the programming paradigm for large-scale data intensive applications". V jeho pojetí jsou NoSQL databáze příliš jednoduché s omezenou funkcionalitou – byly navrženy aplikačními programátory, kteří potřebovali vyřešit svoje konkrétní problémy a nechtěli/nemohli do nich investovat příliš mnoho času. Moderní databáze musí umožňovat snadný a flexibilní přístup k datům (což map/reduce z několika důvodů nesplňuje) a stále moderní databáze musí garantovat kvalitu uložených dat (ACID). Své názory uplatnil při návrhu databáze

SciDB. SciDB je nová databáze, která podporuje funkcionální API a zjednodušené SQL, která podporuje masivní distribuci a paralelismus, která splňuje ACID kritéria, a kde základním databázovým objektem není relace ale pole.

Do Stonebrakerova konceptu jednoúčelových databází zapadají i tzv sloupcové databáze. Tyto databáze jsou navrženy k jedinému účelu – efektivně zpracovat dotazy v OLAP prostředí, kde se uplatňuje schéma star nebo snowflake, kde dominantní operací je SELECT a kde se předpokládá pouze bulk load dat. Tyto databáze jsou relativně nové. K dispozici je několik komerčních řešení – a několik (málo) řešení s otevřeným kódem. Z komerčních řešení je tu databáze Vertica (vlastní HP), jejíž interface (včetně uživatelského) odpovídá PostgreSQL. Můžete si také vyzkoušet databázi GreenPlum, což je komerční fork PostgreSQL umožňující paralelní distribuované zpracování dotazů. Další komerční databází je Ingres/VectorWise od Activianu, což je spojení forku MonetDB s databází Ingres. Z Open Source databází je to především MonetDB. Vyzkoušet si můžete LucidDB, jejíž vývoj ovšem ustrnul na mrtvém bodě.

Na rozdíl od OLTP, kde neexistuje vzor pro návrh schématu, v OLAP databázích se uplatňuje vzor „star“ nebo „snowflake“. Prakticky všechny prezentační nástroje s těmito vzory počítají a umějí je zobrazit, případně umožňují interaktivní analýzy dat. Tyto schémata vyžadují široké faktové tabulky s mnoha desítkami sloupců, přičemž ale většina dotazů vyžaduje data pouze z několika sloupců. Pokud faktovou tabulku uložíme do klasické relační databáze (s uložením dat po řádcích), tak dotazy na několik málo sloupců budou neefektivní – vždy se z disku načítá kompletní řádek – tedy i data, která nepotřebujeme. V klasických databázích je efektivní update řádků – vždy se vejdeme do jedné datové stránky. Ve sloupcových databázích je to přesně naopak – efektivní je čtení – kdy se čtou pouze data sloupců, které jsou nutné pro zpracování dotazu. Naopak neefektivní je update řádků – je nutné načíst a uložit více než jednu datovou stránku (v závislosti na implementaci). U sloupcových databází se předpokládá, že dominantním příkazem bude příkaz SELECT – databáze bude více/méně podporovat pouze čtení – případně přírůstkový masivní zápis. Modifikace nebo přidávání několika málo řádků není časté a tudíž nemusí být řešeno s ohledem na výkon. To je naprosto rozdílná filozofie než ta, která se uplatnila při návrhu klasických SQL databází. Díky tomu poskytují sloupcové databáze řádově rychlejší zpracování SELECTů (případně mnohem menší ztrátu výkonu a další problémy spojené s naléváním dat). Na druhou stranu jednořádkové modifikace databáze jsou většinou řádově pomalejší než u klasických databází. To ale není problém. V konceptu jednoúčelových databází můžeme mít výkonný server zaměřený na transakce (OLTP) např. VoltDB a výkonný server zaměřený na analytiku (OLAP) např. Verticu. Tato kombinace nám může poskytnout řádově větší výkon (na stejném hw) než klasická univerzální databáze bežící v clusteru.

Z pohledu architektury je MonetDB hybridní databáze – kombinace sloupcové a paměťové databáze. Motivací k vývoji této databáze bylo zvyšování ka-

pacit RAM spojené se snižováním ceny. Ve chvíli, kdy nebudeme nuceni při zpracování dat přistupovat na disk, tak se zákonitě hrdlem stane CPU a přístup do pamětí. MonetDB je psán tak, aby SQL zpracovával optimálně z pohledu využití CPU (preferují se jednoduché cykly nad poli) – a to skutečně dělá (za předpokladu, že máme dostatek RAM). Testy, které používáme v GoodData pro hodnocení databázových systémů, zpracovával MonetDB o řád až dva řády rychleji než PostgreSQL.

Např. jednoduchý dotaz `SELECT (age - 30) * 4 FROM tab WHERE age > 30` klasický executor provede ve dvou krocích – jednak se volá sekvenční scan s filtrem, druhak se volá interpret výrazů, kde se pro každý řádek volá funkce o dvou parametrech realizující operátor „mínus“ a funkce o dvou parametrech pro operátor „součin“. Dochází k velkému množství volání podprogramů – navíc téměř vždy se používá interpret výrazů – zpracování výrazů se v relačních databází nekompiluje do strojového kódu (což, pokud je hrdlem IO, nevadí). V MonetDB mají funkce jako parametry pole a výsledkem volání těchto funkcí je pole.

```
<pre>
int age[100] = {20,40,49,33,..};
int aux1[100];
int result[100];
int i;

void
mi(int *p1, int c, int *r, int n)
{
      int i;

      for (i~= 0; i~< n; i++)
      {
            r[i] = p1[i] - c;
      }
}

void
mul(int *p1, int c, int *r, int n)
{
      int i;

      for (i~= 0; i~< n; i++)
      {
            r[i] = p1[i] * c;
```

```
	}
}

	/*
	 * vyhodnoceni vyrazu v~MonetDB
	 */
	mi(age, 30, aux1, 100);
	mul(aux1, 4, result, 100);

	/* vysledek je nyni v~poli result */
</pre>
V~klasické relační db by spíše odpovídal kód
<pre>
int
mi(int a, int b)
{
	return a~- b;
}

int
mul(int a, int b)
{
	return a~* b;
}

	/*
	 * vyhodnoceni vyrazu v~klasicke db - silne zjednoduseno
	 */
	for (i~= 0; i~< 100; i++)
	{
		result[i] = mul(mi(age[i], 30), 4);
	}
</pre>
```

Nevýhodou MonetuDB je jeho paměťová náročnost a chybějící paralelismus. Z pohledu Open Source je škoda, že fork MonetDB – X100, který řešil řadu problémů s náročností na paměť nebyl backportován zpět, ale byl uzavřen a komercializován jako databáze VectorWise (což je naopak pochopitelné z pohledů autorů). Před několika lety byl VectorWise integrován do databáze Ingres (to je krok, který nedokáži pochopit – velice moderní engine byl naroubován do jedné z nejstarších dostupných relačních SQL databází se zastaralým UI). Na rozdíl od LucidDB se MonetDB aktivně vyvíjí – odstraňují se chyby, vydávají se nové

verze. Pro praktického uživatele může být problém akademické zaměření MonetDB – je kladen větší akcent na výzkum (např. implementace podpory polí) než na řešení praktických problémů (paměťová náročnost, chybějící podpora paralelismu).

Sloupcové databáze představují nový a mladý směr vývoje databází, který stojí za to brát do úvahy při návrhu architektury vlastních systémů. Vzhledem k době existence nelze čekat, že nabídnou stejnou funkcionalitu a komfort jako klasické relační databáze, které mají za sebou 30/40 let vývoje – s vývojem sloupcových databází se začalo po roce 2000. Už nyní nabízí zajímavý výkon, který muže řešit reálné výkonostní problémy aniž by uživatel musel investovat do komplexnější nebo výkonější infrastruktury.

Proti proudu času aneb Oracle flashback technologie?

Petr Jiroušek

E-mail: petr@civ.zcu.cz

Stejně tak jako i v jiných technických oblastech i v oblasti databází a na nich založených informačních systémech hraje důležitou úlohu člověk. A člověk, jak známo, je bytost omylná. A tak je nutné s touto eventualitou počítat i na tomto poli. Proti tomu, že uživatel nebo administrátor databáze svou nepozorností nebo omylem zadá chybná data nebo provede chybnou operaci se lze bránit poměrně jednoduše. Lze k tomu využít něco, co dnes nabízí každá databáze... a to pravidelné zálohování. Pokud máme všechna data řádně zazálohována, pak prakticky všechny běžně používané databáze dnes nabízejí tzv. „point-in-time recovery“, čímž obnovíme všechna data k libovolnému konkrétnímu okamžiku v minulosti. Z hlediska záchrany dat je tato metoda výborná, kdo však takovýmto úkolem nebývá příliš nadšen, jsou databázoví administrátoři. Pokud jde o provozní systém, pak je totiž většinou potřeba další hardware, na kterém se obnovení původní databáze provede, ale hlavně pak je potřeba ona mravenčí práce k „vykuchání“ konkrétních dat, které potřebujeme opravit a jejich přenesení do provozního systému, posouzení všech případných návazností a finálně vlastní obnova. Lze najít i případy, kdy bohužel ani touto metodou nelze požadovaná data obnovit do původního stavu. Jedná se zejména o ty případy, kdy jsou vzájemné vazby mezi daty natolik složité, že potřeba obnovit data v jedné tabulce vyvolá potřebu obnovit data v mnoha a mnoha dalších tabulkách. V takových případech se pak většinou rozhodujeme pro jiná řešení, buď tedy chybu neopravovat vůbec, nebo pro obnovu celé databáze k okamžiku „před chybou“ a zahodíme tak případné další změny, které byly od té doby v databázi provedeny. Zde už pak většinou nerozhodují možnosti technické ale spíše finančně manažerské...

Databáze Oracle v tomto směru přinesla na konci minulého století novinku tzv. flashback technologie. Na úvod to byla technologie Flashback Query. Její základní použití si ukážeme na následujícím příkladu. Použijeme tabulku EMPLOYEES (standardní součást testovacích tabulek v databázi Oracle), ze které pro přehlednost zrušíme některé sloupečky a část testovacích dat. Zde je počáteční stav dat v naší tabulce:

```
14:24:58 SQL> select employee_id,first_name,last_name,phone_number from
employees;

EMPLOYEE_ID FIRST_NAME      LAST_NAME       PHONE_NUMBER
----------- --------------- --------------- ---------------
110         John            Chen            515.124.4269
111         Ismael          Sciarra         515.124.4369
112         Jose Manuel     Urman           515.124.4469
114         Den             Raphaely        515.127.4561
115         Alexander       Khoo            515.127.4562
116         Shelli          Baida           515.127.4563
118         Guy             Himuro          515.127.4565
119         Karen           Colmenares      515.127.4566
120         Matthew         Weiss           650.123.1234

9 řádků vybráno.
```

Provedeme v této tabulce několik změn a celou transakci potvrdíme příkazem commit:

```
14:25:01 SQL> update employees set phone_number='111.111.1111' where
employee_id=120;

1 řádek aktualizován.

14:25:12 SQL> update employees set first_name='Jerry' where employee_id=110;

1 řádek aktualizován.

14:25:22 SQL> delete from employees where employee_id=114;

1 řádek odstraněn.

14:25:32 SQL> insert into employees values(121,'John','Dee','234.333.3424');

1 řádek vytvořen.

14:26:12 SQL> commit;

Potvrzení dokončeno.
```

A nyní si opět vypíšeme obsah tabulky EMPLOYEES:

```
14:26:16 SQL> select employee_id,first_name,last_name,phone_number from
employees;

EMPLOYEE_ID FIRST_NAME      LAST_NAME       PHONE_NUMBER

----------- --------------- --------------- ---------------
110         Jerry           Chen            515.124.4269
111         Ismael          Sciarra         515.124.4369
112         Jose Manuel     Urman           515.124.4469
115         Alexander       Khoo            515.127.4562
```

```
116         Shelli          Baida           515.127.4563
118         Guy             Himuro          515.127.4565
119         Karen           Colmenares      515.127.4566
120         Matthew         Weiss           111.111.1111
121         John            Dee             234.333.3424

9 řádků vybráno.
```

Vše dopadlo tak, jak jsme předpokládali Všechny námi provedené změny jsou v databázi uloženy. Změnili jsme telefonní čísla u zaměstnanců s `ID = 110` a `120`, vymazali jsme zaměstnance s `ID = 114` a vložili nového zaměstnance s `ID = 121`.

Pro demonstraci flashback technologie přidáme k výše uvedenému příkazu select „magickou“ klauzuli AS OF TIMESTAMP. Tato klauzule zapíná tzv. Flashback Query a při splnění určitých podmínek, které si popíšeme níže, tímto příkazem umíme vypsat data z tabulky tak, jak tam byla uložena v minulosti. V našem případě budeme chtít data vypsat tak, jak byla v tabulce uložena před jednou minutou. Použijeme k tomu konstrukci jazyka SQL: `SYSTIMESTAMP - INTERVAL '1' MINUTE`

```
14:26:54 SQL> select employee_id,first_name,last_name,phone_number from
employees as of timestamp systimestamp-interval '1' minute;

EMPLOYEE_ID FIRST_NAME      LAST_NAME       PHONE_NUMBER

----------- --------------- --------------- ---------------
110         John            Chen            515.124.4269
111         Ismael          Sciarra         515.124.4369
112         Jose Manuel     Urman           515.124.4469
114         Den             Raphaely        515.127.4561
115         Alexander       Khoo            515.127.4562
116         Shelli          Baida           515.127.4563
118         Guy             Himuro          515.127.4565
119         Karen           Colmenares      515.127.4566
120         Matthew         Weiss           650.123.1234

9 řádků vybráno.
```

A opět dle očekávání vidíme opravdu stav dat tak, jak byl v tabulce před jednou minutou. Veškeré změny jakoby se vůbec neudály. Pomocí následujícího příkazu pak máme dokonce možnost prohlédnout si detailně historii jednotlivých řádků v tabulce EMPLOYEES:

```
14:27:17 SQL> select employee_id,versions_starttime, versions_endtime,
versions_xid, versions_operation from employees versions between timestamp
minvalue and maxvalue order by employee_id;

EMPLOYEE_ID VERSIONS_STARTTIME  VERSIONS_ENDTIME    VERSIONS_XID     VER
```

```
---------- ----------------- ----------------- ---------------- ---
110                          02.09.12 14:26:14
110        02.09.12 14:26:14                   06000F0010030000 U
111
112
114        02.09.12 14:26:14                   06000F0010030000 D
114                          02.09.12 14:26:14
115
116
118
119
120        02.09.12 14:26:14                   06000F0010030000 U
120                          02.09.12 14:26:14
121        02.09.12 14:26:14                   06000F0010030000 I

13 řádků vybráno.
```

Jednotlivé řádky jsou identifikovány příslušným ID zaměstnance v prvním sloupečku. U všech záznamů, se kterými bylo manipulováno, vidíme čas, do kdy platila předchozí verze dat, od kdy platí nová verze dat (což je logicky úplně stejný čas) a též vidíme typ a interní identifikátor transakce, která data zmodifikovala.

Pokud bychom chtěli obnovit data v tabulce k nějakému času v minulosti, opět nás čeká trocha ruční práce. Automaticky na „jedno kliknutí“ to ani v tomto případě nejde. Opět musí člověk posoudit, které data přesně obnovit, zda tím jsou splněny všechny případné návaznosti atd. Ale pokud je mezi čtenáři někdo, kdo někdy zkoušel obnovu databáze standardními metodami, pak mi zcela jistě dá za pravdu, že v tomto případě se jedná o významnou úsporu času, práce i potřebných znalostí.

Pohled do střev ...

Předpokládám, že se najdou zvědavci, kterým vrtá v hlavě, jakým způsobem tato technologie funguje. Takže se to v následující části pokusím nastínit. Základem všeho je ona tzv. transakce. Její asi nejdůležitější vlastností je to, že dokud ji nepotvrdíme (příkazem commit), všechny operace, které děláme, si děláme takříkajíc na vlastní písečku. Kdokoli jiný, kdo v tu chvíli v databázi pracuje vidí všechna data taková, jaká byla předtím, než jsme naši transakci odstartovali. Je tedy patrné, že databáze si někde na zvláštním místě musí ukládat po celou dobu, kdy naši transakci provádíme všechny dvojice stará – nová data a v okamžiku potvrzení transakce pak z tohoto místa musí ostrá data přepsat novými hodnotami. V terminologii Oracle se toto zvláštní místo nazývá „undo“. Fyzicky se jedná o speciální soubor, který je stejně jako všechny datové soubory uložen kdesi na souborovém systému. V tomto souboru si vždy každá transakce alokuje určité místo na ona stará a nová data a po ukončení transakce alokované

místo opět uvolňuje. Pokud místo v souboru dochází, pak se souboru může automaticky nafukovat až po mez, kterou buď určí administrátor databáze nebo operační systém. Tedy tak by tomu bylo, pokud bychom nepoužívali technologii Flashback Query. V tomto případě je management alokace poněkud modifikován a tato undo data zůstávají v souboru ještě nějakou dobu i po potvrzení transakce. Záměrně píši nějakou, neboť tato doba je pro každou databázi zcela individuální. Standardně je přednastaveno přesně 15 minut ale v tzv. režimu bez garance. Znamená to, že minimálně po dobu 15 minut budou pro každou transakci příslušná undo data k dispozici, pokud nedojde k problémům s alokací místa pro nové transakce. Problémy je v tomto případě myšleno zejména to, že nebude možno z libovolného důvodu soubor zvětšit a nové transakce budou chtít alokovat více místa, než je momentálně k dispozici. Pak dojde k přepisování těchto undo dat z již ukončených transakcí i před 15 minutou. Administrátor teoreticky může nastavit garanci těchto 15 minut, pak by ale nové transakce nemohly být odstartovány, dokud těchto 15 minut nevyprší. Toto nastavení se však v praxi příliš nepoužívá. Vzhledem k tomu, že algoritmus managementu undo souboru se nejprve snaží obsadit prázdné bloky v souboru a teprve pokud žádné nenajde se snaží obsazovat nejstarší undo bloky potvrzených transakcí, nastává daleko častěji situace, že můžeme pomocí Flashback Query obnovit data stará i několik dnů či týdnů. Vše záleží zejména na tom, kolik a jak velkých transakcí běžně v databázi probíhá a v kterém okamžiku tudíž nastane stav, že místo v undo souboru opravdu dojde a začnou se přepisovat data nejstarších transakcí.

Další vývoj flashback technologií

Vzhledem k velkému úspěchu této technologie se postupně začaly objevovat další funkce této rodiny flashback technologií. Nejprve se tato funkčnost pouze zdokonalovala a přibývaly další parametry, podle kterých bylo možné vyhledávat historická data. V další fázi se v databázi objevil klasický odpadkový koš ze kterého bylo možné obnovit všechny objekty, které jsme v databázi smazali Tato technologie se tedy netýká vlastních dat. Ale stále bylo limitující to, že veškerá funkčnost závisela na undo datech, která časem prostě zmizela. Proto byla vyvinuta další skupina flashback technologií, která využívá speciální tzv. Flashback Logy. Tam si databáze ukládá ve speciálním formátu všechny operace, které se v databázi staly. Jako první z této rodiny byla implementována funkce Flashback Database, která jednoduše celou databázi přesune zpět do specifikovaného časového okamžiku. Příslušný příkaz vypadá následovně:

```
FLASHBACK DATABASE TO TIMESTAMP timestamp'2002-11-05 14:00:00';
```

Po provedení tohoto příkazu „zmizí z databáze“ všechny operace, které byly potvrzeny až po tomto čase. Tato technologie se následně začala také využívat

v testovacích procesech, kdy je potřeba provést více testů se shodnými počátečními podmínkami.

Postupně se však ukázalo, že větší poptávka je po funkcích, které umožňují prohledávat historická data. Proto v další verzi Oracle databáze (11g) byla technologie Flashback Query kompletně přepracována tak, že u provozních tabulek, které si určíme, jsou všechna historická data ukládána do standardního databázového souboru na disku ve formě tabulek se shodnou strukturou, jako mají provozní tabulky. To zjednodušuje vlastní vyhledávání. Postačí nám pouze znát základní syntaxi jazyka SQL a pak způsob, jakým Oracle pojmenovává tyto historické tabulky, a můžeme se vesele pustit proti proudu času. Tato technologie byla pojmenována nejprve Flashback Data Archive, později byla pak přejmenována na Oracle Total Recall. My si nejprve vytvoříme databázový soubor pro ukládání Flashback Archivu:

```
CREATE TABLESPACE FLASH DATAFILE '/mnt/data/oracle/orcl/flash.dbf' SIZE 100M
AUTOEXTEND ON MAXSIZE UNLIMITED;
```

Vlastní flashback archiv pak vytvoříme příkazem:

```
CREATE FLASHBACK ARCHIVE DEFAULT fla1 TABLESPACE flash RETENTION 1 YEAR;
```

U provozních tabulek, které chceme mít ve Flashback Archivu pak použijeme příkaz:

```
ALTER TABLE employee FLASHBACK ARCHIVE fla1;
```

A pak již stačí mít jen na diskovém prostoru dostatek místa. Tímto se zbavíme závislosti na undo datech a hlavně na jejich základní vlastnosti, tj. jejich postupném přepisování. Tato komplexní technologie je využívána již nejen k prohlížení historických dat a jejich případné obnově, ale také může fungovat i jako audit všech operací nad konkrétními daty. Pro ukázku této technologie opět použijeme tabulku EMPLOYEES. Její aktuální stav po našem prvním příkladu je následující:

```
SQL> select * from employees;

EMPLOYEE_ID FIRST_NAME      LAST_NAME       PHONE_NUMBER

----------- --------------- --------------- ------------
110 Jerry           Chen            515.124.4269
111 Ismael          Sciarra         515.124.4369
112 Jose Manuel     Urman           515.124.4469
115 Alexander       Khoo            515.127.4562
116 Shelli          Baida           515.127.4563
118 Guy             Himuro          515.127.4565
119 Karen           Colmenares      515.127.4566
120 Matthew         Weiss           111.111.1111
121 John            Dee             234.333.3424

9 řádků vybráno.
```

Provedeme opět několik změn v datech:

```
SQL> delete from employees where employee_id=115;

1 řádek odstraněn.

SQL> insert into employees values(122,'Alex','Smith','333.333.3333');

1 řádek vytvořen.

SQL> update employees set phone_number='098.765.4321' where mployee_id=118;

1 řádek aktualizován.

SQL> delete from employees where employee_id=121;

1 řádek odstraněn.

SQL> insert into employees values(123,'Johnie','Walker','777.666.5555');

1 řádek vytvořen.

SQL> update employees set phone_number='222.333.4321' where employee_id=116;

1 řádek aktualizován.

SQL> commit;

Potvrzení dokončeno.
```

Nyní si najdeme název tabulky, ve které jsou všechna historická data uložena:

```
SQL> select archive_table_name from user_flashback_archive_tables where
table_name='EMPLOYEES';

ARCHIVE_TABLE_NAME

--------------------------------------------------
SYS_FBA_HIST_165854
```

A následně si jednoduchým příkazem select můžeme prohlédnout kompletní historii dat v naší tabulce:

```
SQL> select startscn,endscn,operation,employee_id,first_name,last_name,
phone_number from SYS_FBA_HIST_165854;

   STARTSCN      ENDSCN O EMPLOYEE_ID FIRST_NAME      LAST_NAME      PHONE_NUMBER

----------- ----------- - ----------- --------------- -------------- ------------
48183829568 48183829568 D         115 Alexander       Khoo           515.127.4562
            48183829568           115 Alexander       Khoo           515.127.4562
            48183829568           116 Shelli          Baida          515.127.4563
            48183829568           118 Guy             Himuro         515.127.4565
```

```
48183829568 48183829568 D         121 John            Dee             234.333.3424
            48183829568           121 John            Dee             234.333.3424

6 řádků vybráno.
```

Ve výpisu jsou všechna historická data plus jejich historie, tj. všechna data, se kterými bylo manipulováno zde mají jeden řádek, který začíná s prázdným STARTSCN (SCN je časová značka databáze) a končí konkrétním SCN, kdy změna proběhla. U smazaných záznamů je zde ještě zvláštní řádek s posledními hodnotami, které v databázi byly uloženy. Pokud někoho zaráží, že zde nejsou vidět operace vkládání nových záznamů, pak je to v pořádku. Vložené záznamy jsou přímo v tabulce EMPLOYEES a dokud s nimi neprovedeme žádnou operaci, pak se mezi historickými daty o nich žádná zmínka neobjeví. Můžeme to experimentálně ověřit tím, že provedeme změnu některého nově vloženého záznamu:

```
SQL> update employees set phone_number='444.444.4444' where employee_id=123;

1 řádek aktualizován.

SQL> commit;

Potvrzení dokončeno.
```

A následně si opět vypíšeme obsah Flashback Archivu:

```
SQL> select startscn,endscn,operation,employee_id,first_name,last_name,
phone_number from SYS_FBA_HIST_165854;

   STARTSCN      ENDSCN O EMPLOYEE_ID FIRST_NAME      LAST_NAME      PHONE_NUMBER

----------- ----------- - ----------- --------------- -------------- ------------
48183839933 48183839933 D         115 Alexander       Khoo           515.127.4562
            48183839933           115 Alexander       Khoo           515.127.4562
            48183839933           116 Shelli          Baida          515.127.4563
            48183839933           118 Guy             Himuro         515.127.4565
48183839933 48183839933 D         121 John            Dee            234.333.3424
            48183839933           121 John            Dee            234.333.3424
48183839933 48183839936 I         123 Johnie          Walker         777.666.5555

7 řádků vybráno.
```

Vidíme, že hodnoty z příkazu pro vložení záznamu jsou již ve Flashback Archivu uložené, aktuální stav s novým telefonním číslem je opět pouze v tabulce EMPLOYEES.

Pro případné zájemce je ale nutné poznamenat, že kromě původní Flashback Query technologie jsou všechny ostatní flashback technologie dostupné pouze v nejdražší edici Oracle databáze – v tzv. Enterprise Edition.

Více informací lze nalézt zejména na:
http://docs.oracle.com/cd/E11882_01/appdev.112/e25518/adfns_flashback.htm

Vývoj aplikací v Qt pro mobilní zařízení

Jozef Mlích, Aleš Láník, Pavel Zemčík

E-mail: {imlich,ilanik,zemcik}@fit.vutbr.cz

Abstrakt

Tento článek popisuje možnosti vývoje aplikací v Qt na různých mobilnícn platformách. Jsou zde shrnuty některé pojmy týkající se jednotlivých platforem. Dále jsou zde diskutovány jednolivé aspekty vývoje aplikací pro patformy MeeGo Harmattan, Maemo Fremantle a Symbian.

1 Úvod

Mobilní telefony v posledním desetiletí prošly velkým vývojem. Firma Nokia v roce 2001 koupila firmu *Trolltech*, později *Qt Software*, aby mohla vyvíjet software pro mobilní telefony. Tento toolkit byl postupně přizpůsoben pro tehdy používaný systém *Symbian*. Dále byl tento nástroj přizpůsoben pro platformu *Maemo Fremantle* a jejího nástupce *MeeGo Harmattan*.

Firmu *Qt software technologies*, která vyvíjí *Qt*, koupila firma *Digia*, která přislíbila podporu Qt na systémech Android a iPhone. Qt je multiplatformní framework pro vývoj aplikací. Obsahuje velké množství komponent pro vývoj grafických uživatelských rozhraní [5]. Od verze 4.7 obsahuje skriptovací jazyk *QML* pro vytváření uživatelských rozhraní. Pro mobilní telefony byla v tomto jazyce vytvořena sada grafických prvků tzv. *Qt Components* přizpůsobených pro mobilní telefony. Knihovna *Qt* zapouzdřuje starší API systému *Symbian* [4].

V první části toho článku jsou popsány vlastnosti systému *MeeGo*, který je v kontextu tohoto článku brán jako referenční mobilní platforma. Ve druhé části se článek zabývá odlišnostmi a systému *Symbian* od *MeeGo* z pohledu vytváření aplikací v *Qt*. V poslední části jsou shrnuty odlišnosti programování aplikací v systému *Maemo*.

2 MeeGo

MeeGo je operační systém založený na Linuxu určený pro mobilní platformy. Vznikl na základě spolupráce celé řady firem, zejména firmy Nokia a Intel. Základem systému je Linuxové jádro a další základní komponenty využívané na stolních počítačích v Linuxu. Přínosem systému jako takového je sada knihoven potřebných pro mobilní zařízení, počítače vestavěné do aut, počítače vestavěné do televizí, tablety nebo netbooky. Firma Nokia se spíše zaměřovala na využití v mobilních telefonech. Firmy okolo Intelu se zaměřili na vývoj obecné platformy pro ostatní uvedené typy zařízení. Dalším přínosem je uživatelské rozhraní přizpůsobené pro využití systému v těchto zařízeních.

Existují tedy dvě varianty systému *MeeGo*. První varianta je potomkem systému *Moblin*. Používá balíčkovací systém *RPM* a verze určená pro mobilní telefony je označovaná jako *MeeGo Handset UX*. Na základě této verze vznikl *Mer project*[1] a uživatelské rozhraní je pojmenováno *Nemo mobile*.

Druhá varianta tohoto systému je označována jako *MeeGo Harmattan*, která vychází z *Maemo 5 Fremantle*. Hlavní rozdíly jsou v použitém balíčkovacím systému a uživatelském rozhraní *Swipe*. Při srovnání obou variant je vidět, že obě využívají velké množství běžných linuxových knihoven. Je tedy možné využít velké části aplikací nebo celé aplikace vytvořené pro stolní počítače nebo obecně pro Linux.

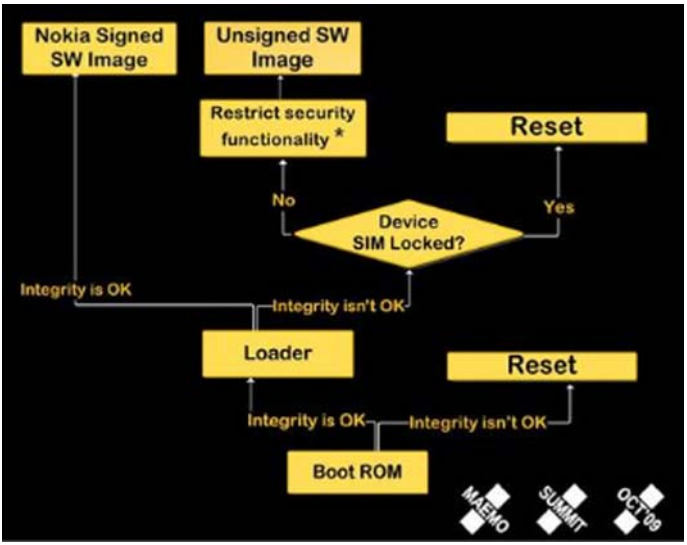
2.1 Aegis

Jedním z významných rozdílů systému *MeeGo Harmattan* ve srovnání s ostatními linuxovými systémy je zabezpečení systému s využitím rozšíření systému nazvaného *Aegis*. Toto rozšíření zjišťuje kontrolu důležitých systémových součástí a jednotlivých nainstalovaných aplikací. U jednotlivých souboru se ověřuje jejich konzistence, přičemž v případě nesprávného kontrolního součtu se zamezí jeho spuštění, případně spuštění samotného zařízení. Tento systém umožňuje omezit přístup k určitým zdrojům telefonu. Obvykle používané POSIXové omezení oprávnění pro toto řízení přístupu není zřejmě vhodné.

U každého spustitelného programu je definovaný seznam požadovaných oprávnění. Z běžně dostupných oprávnění lze jmenovat například možnost odeslání SMS zprávy, manipulace s touto zprávou, přístup k informacím ze sociálních sítí, přístup k informacím o poloze a natočení zařízení, atd.

Pro získání přístupu k těmto službám je nutné do instalačního balíčku vložit soubor *aegis.manifest*. Při použití vývojového prostředí Qt Creator jsou soubory potřebné k vytvoření balíčku a nastavení oprávnění součástí šablony „hello world“ programu. Pro přidání požadavku na příslušné oprávnění stačí vytvořit

[1] http://www.merproject.org/



Obr. 1: Ověřování bezpečnosti software pomocí programu Aegis

program, který toto oprávnění vyžaduje a vývojové prostředí přidá tento požadavek do manifestu. Toho lze dosáhnout například vložením textu „`import QtMobility.location 1.2`“ do zdrojového kódu v jazyce QML.

Příklad 1: Soubor aegis.manifest pro program /opt/foo/bin/foo

```
<aegis>
    <request policy="add">
        <credential name="FacebookSocial" />
        <credential name="TrackerReadAccess" />
        <credential name="TrackerWriteAccess" />
        <credential name="Location" />
        <credential name="GRP::pulse-access" />
        <credential name="GRP::video" />
        <credential name="GRP::audio" />
    </request>
    <for path="/opt/foo/bin/foo" />
    <for path="applauncherd-launcher::/usr/bin/applauncherd.bin" id="" />
</aegis>
```

V některých případech, například pro přístup k MMS, se používá AegisFS. Jedná se o souborový systém využívající FUSE[2]. AegisFS připojí do daného adresáře část filesystému a umožní z něj číst pouze vybraným procesům.

[2] filesystem in user space

Pro získání přístupu ke všem oprávněním včetně systémových se musí nainstalovat balíček *Inception*[3]. Jedná se například o oprávnění, které umožňují změnu vzhledu základních nabídek systému. Příkazem `accli -I` v terminálu lze zjistit seznam dostupných oprávnění.

2.2 Software

V případě systému *MeeGo Harmattan* lze instalovat software z několika zdrojů. První možností je *Nokia Store*, který je jediný povolený ve výchozím nastavení.

Druhou možností pro instalaci aplikací je instalace s využitím repositáře apps.formeego.com. Jedná se o komunitní repozitář, kde probíhá schvalování jednotlivých aplikací na základě recenzí uživatelů. Pro instalaci z tohoto repositář existuje program s GUI.

Další možností je instalovat software bez použití repozitářů přímo ze souboru balíčku (obvykle soubor s příponou `.deb`). Tato varianta není vhodná s ohledem na další aktualizace programu ani bezpečnost. Při vkládání balíčku do repozitáře se obvykle ověřuje kdo je autorem balíčku. Poslední možností je použít další repositáře. Pro instalaci software z těchto repositářů momentálně neexistuje program s GUI. V případě použití těchto alternativních repositářů může docházet k různým potížím. Jejich seznam lze nalézt na N950 langing page[4].

3 Symbian

Z pohledu vývoje aplikací v *Qt* a *QML* lze brát *MeeGo Harmattan* jako referenční platformu. *Qt SDK* je vhodné pro vývoj aplikací pro *Symbian Anna* a *Symbian Belle*. Ostatní platformy nejsou doporučeny zejména kvůli nedostatečnému výpočetnímu výkonu zařízení [3].

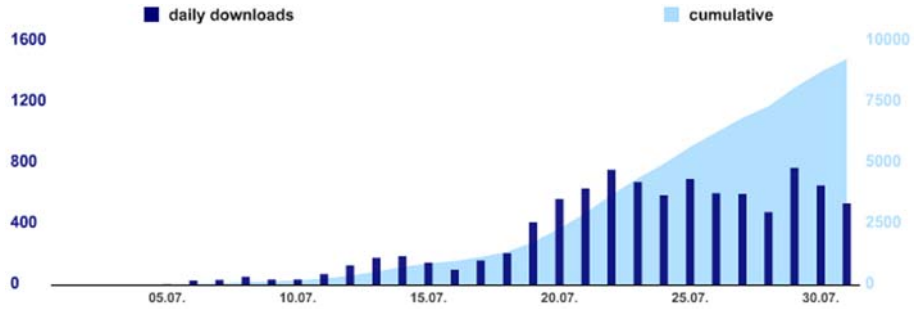
Podle oficiálních informací je v *Nokia Store* [1] je celkově více než 120 000 aplikací. Podle statistik pro Český trh získaných z aplikace czMeteo (viz Obrázek 2) lze říci, že zařízení se systémem *Symbianem Anna* nebo *Symbian Belle* je přibližně 10 krát více než zařízení se systémem *MeeGo Harmattan*. Největší podíl mají zařízení Nokia N8, což přibližně odpovídá celosvětovému podílu.

3.1 Projekt v Qt

Pro vývoj v *Qt* pro *Symbian* je nutné nainstalovat *Qt SDK*. Toto SDK existuje ve verzi pro Windows a pro Linux. Verze pro Windows obsahuje všechny potřebné nástroje pro vývoj aplikací pro Symbian. Některé z těchto nástrojů nejsou dostupné v Linuxové verzi. Implementace jednotlivých knihoven Qt (Qt Mobility

[3] viz https://endno.de/ itsnotabigtruck/inception/

[4] viz http://wiki.meego.com/N950_landing_page



Obr. 2: Statistiky stažení aplikace czMeteo za červenec 2012

Location, Qt Mobility Messaging, atd.) jsou většinou obálkou (tzv. wrapper) nad staršími knihovnami pro *Symbian*.

Základní projekt se vytvoří z šablon obsažených ve vývojovém prostředí. Při překladu pro *Symbian* se vygeneruje sis balíček, který se nainstaluje do zařízení přes RS 232 nebo přes USB. Na telefonu musí být nainstalovaný software CODA. Jedná se o zkratku *Carbide.C++ On-device Debugging Agent*. Tento software umožňuje spojení programu s vývojovým prostředím pro ladění programu. Pro vývoj software lze také použít i starší TRK. Jeho použití se však v současnosti nedoporučuje.

Pro překlad programu a vytvoření balíčku pro *Symbian* na Linuxu je situace složitější. Je možné nainstalovat si *Symbian Qt SDK for Linux*, který není oficiálně podporován. Další možností je použít *Remote Compiler*, kdy se pro vytvoření balíčku celý zdroják pošle na vzdálený server a tam se přeloží zvolenou verzí *Qt*, vyrobí se příslušný balíček, pošle zpět a následně se zobrazí chyby při překladu.

Při vytváření balíčku pro *Symbian* je potřeba upravit projekt (v souboru s příponou `.pro`) tak, aby v něm byly uvedeny potřebné závislosti (`TARGET.CAPABILITY` a `CONFIG += qt-components`). Místo `.desktop` souboru s názvem programu, jeho ikonou a popisem, používaného obvykle v Linuxu pro vytvoření položky v menu nebo na ploše, je potřeba vložit tyto údaje do projektového souboru (`icon`, `version` a `deployment.display_name`). Dále je zde položka `TARGET.UID3`, která obsahuje unikátní ID aplikace. Pro účely testování je vhodné použít předem vygenerované `UID`. Ukázka části projektového souboru je v následujícím příkladu. Položka `EPOCHEAPSIZE` definuje velikost haldy (heap) potřebné pro spuštění aplikace.

Příklad 2: část projektového souboru specifická pro Symbian

```
symbian {
        TARGET = "foo_FFFFFFFF"
```

```
        TARGET.UID3 = 0xFFFFFFFF
        TARGET.EPOCHEAPSIZE = 0x20000 0x2000000
        ICON = "foo.svg"
        VERSION = 0.0.1
        DEPLOYMENT.display_name = "foo bar"
        TARGET.CAPABILITY += NetworkServices Location
        TARGET.CAPABILITY += SwEvent
        my_deployment.pkg_prerules += vendorinfo

        my_deployment.pkg_prerules += \
                "; Dependency to Symbian Qt Quick components" \
                "(0x200346DE), 1, 0, 0, {\"Qt Quick components\"}"
        DEPLOYMENT += my_deployment
        vendorinfo += "%{\"xmlich02\"}" ":\"xmlich02\""
}
CONFIG += qt-components
```

Při instalaci aplikace je nutné se zabývat i tím zda je na zařízení *Qt* vůbec nainstalované a v jaké verzi. Toto závisí na typu zařízení. Pro doinstalování potřebné verze *Qt* se při vytváření balíčku zaškrtne volba „Smart Installer“, který při prvním spuštění doinstaluje potřebné knihovny. Položky s názvem `DEPLOYMENT` popisují závislost na Qt. V tomto případě se jedná o workaround pro chybu v Qt SQK [2]. Pro ladění aplikace je ovšem naopak vhodné Smart Installer po prvním nainstalování do balíčku nepřidávat, protože potom se ladící výpisy nemusí dostat do vývojového prostředí.

3.2 Rozdíly v Qt Components

Vývoj programu je pro *Symbian* z velké části shodný s vývojem pro *MeeGo Harmattan*. Nejvíc rozdílů je asi v `qt-components`. Tyto komponenty jsou v rozdílných jmenných prostorech (namespace). V dalším popisu jsou uvedeny kousky zdrojového kódu, přičemž části pro *MeeGo Harmattan* jsou označeny písmenem `(M)` a části pro *Symbian* jsou označeny písmenem `(S)`.

Příklad 3: Rozdílné jmenné prostory v QML

```
(M) import com.nokia.meego 1.0
(S) import com.nokia.symbian 1.1
```

Součástí komponenty `ToolBarLayout` pro vytváření jednotlivých nabídek jsou tlačítka. Ta jsou v *Symbianu* pojmenovány jinak.

Příklad 4: ToolIcon a ToolButton

```
(M) ToolIcon { platformIconId: "toolbar-back" }
(S) ToolButton { iconSource: "toolbar-back" }
```

Systém *MeeGo Harmattan* používá pro ukončení aplikace gesto, kdy se přejede přes okraj displeje. Ukončovací tlačítko tedy není nutné implementovat. V aplikaci pro *Symbian* je nutné vhodně přidat ukončovací tlačítko.

V systému *Symbian* mají komponenty nastavené tmavý styl jako výchozí. V případě, že má telefon AMOLED displej to má pozitivní vliv na spotřebu energie. Pomocí následujícího kódu lze přepnout styl aplikace do tmavého grafického stylu.

Příklad 5: Nastavení tmavého vzhledu v MeeGo Harmattan

```
Component.onCompleted: {
    theme.inverted = true;
}
```

Na rozdíl od *Symbianu MeeGo* umožňuje u jednotlivých komponent vybrat tmavý styl.

Příklad 6: inverted platformStyle

```
(M) Button { platformStyle: ButtonStyle { inverted: true } }
(S) Button {
  /* platformStyle: ButtonStyle { inverted: true } */
}
```

Součástí vizuálního stylu je i velikost objektu `BusyIndicator`. Tuto velikost lze nastavovat buď pomocí proměnné `platformStyle`, nebo pomocí proměnných `height` a `width`.

Příklad 7: neexistující BusyIndicatorStyle

```
BusyIndicator {
  (M) platformStyle: BusyIndicatorStyle { size: "large" }
  (S) width: 70; height: 70;
}
```

Dále v systému *Symbian* není možné používat další rozšiřující knihovny jako je například *Meego Touch Framework*, *gstreamer* nebo *libcontentaction*. Tyto knihovny nabízí funkce, které v mobilních telefonech často požadovány. Jedná se například o funkce jako „sdílet na facebook“ (viz `shareuiinterface-maemo-meegotouch`). Spuštění vestavěné navigace lze provést tímto voláním:

Příklad 8: Neexistence libcontentaction

```
(M) Qt.openUrlExternally("geo:"+model.lat+","+model.lng)
(S) Qt.openUrlExternally("http://m.ovi.me/?c="+model.lat+","+model.lng);
```

Komponenty `Sheet` a `CountBubble` nemají v *Symbianu* své ekvivalenty. Další komponentou, která má jiné metody je `InfoBanner`. Příklad zdrojového kódu je uveden níže:

Příklad 9: Infobanner má jiné metody

```
InfoBanner { id: banner; }
(S) banner.open()
(M) banner.show()
```

3.3 Ladění a publikování aplikací

Jak už bylo zmíněno, tak pro ladění na fyzickém zařízení se používá CODA nebo TRK. Pro testování aplikace na více různých zařízeních lze využít službu Remote Device Access[5]. Pomocí webové stránky se zvolí daný typ zařízení a rezervuje se na něm potřebný čas. Toto zařízení je následně zpřístupněno přes jws[6] aplikaci. Tato aplikace zpřístupňuje obrazovku zařízení pomocí protokolu VNC. Dále umožňuje instalovat balíčky, přenášet soubory a restartovat zařízení.

Pro ladění aplikací je možné použít i simulátor integrovaný do vývojového prostředí *Qt Creator*. Simulátor je nutné přepnout na *Symbian*. Toto se provede v simulátoru v menu *Application* se v *Choose com.nokia.extras platform* zvolí varianta *Symbian*.

Každá aplikace v *Nokia Store* je označena unikátním id. Pro publikování aplikace je potřeba pro každou aplikaci emailem požádat o takovéto id `UID`. S tímto id jsou svázány i oprávnění poskytnuté dané aplikaci. Seznam oprávnění nutných pro běh aplikace je k dispozici v proměnné `TARGET.CAPABILITY` v projektovém souboru. S výchozím oprávnění například není možné přistupovat k pozici přístroje, kameře, atd. Pro vytvoření aplikace vyžadující některé z vyšších oprávnění nutné požádat o certifikát, který je vystavený dle seznamu IMEI jednotlivých telefonů používaných pro vývoj aplikací.

4 Maemo Fremantle

Systém *Maemo Fremantle* je předchůdcem systému *MeeGo Harmattan*. Tento systém je k dispozici na telefonu *Nokia N900*. Jeho grafické uživatelské rozhraní je založeno na `libhildon`, která vychází z knihoven *Gtk*. V tomto systému je možné vyvíjet aplikace i v *Qt* [6]. Jedním z hlavních problémů jsou různé verze Qt a Qt Components v systému *Maemo*. Tento systém po přeflashování a aktualizaci obsahuje potřebné repositáře extras a příslušné balíčky. Po přidání extras-testing a extras-devel a instalaci Community Updates (CSSU) je to ale možné. Jedná se o balíčky *qt-components-10 libqtm-12 qtquickcompat libqtm-12-location libqtm-11-location qt4-declarative-qmlviewer*.

Tyto balíčky je možné dát do závislostí aplikace. Jak bylo zmíněno výše, tak systém bez rozšíření obsahuje starší verzi *Qt Components*, kde je potřeba přepsat importy, a to prakticky ve všech QML souborech. Novější verze je již kompatibilní s *MeeGo*. Ve zdrojovém kódu pro *MeeGo Harmattan* jsou například tyto importy:

[5] viz též http://rda.cellulardata.com/
[6] Java Web Start

Příklad 10: import jmenných prostorů pro MeeGo Harmattan

```
import Qt.Quick 1.0
import com.nokia.meego 1.1
import com.nokia.extras 1.1
```

Korespondující zdrojový kódu pro *Maemo Fremantle* bude vypadat následovně:

Příklad 11: import jmenných prostorů pro Maemo Fremantle

```
import Qt 4.7
import Qt.labs.components.native 1.0
import org.maemo.fremantle 1.0
import org.maemo.extras 1.0
```

Qt Mobility (tedy knihovna pro práci s gps, akcelerometry apod.) má jiné umístění a název, takže je nutné v projektovém souboru (*.pro) přidat podmíněný překlad pro Maemo:

Příklad 12: Projektový soubor – podmíněný překlad pro Maemo

```
maemo5 {
CONFIG += mobility12
} else {
CONFIG += mobility
}
```

Knihovny Qt Mobility nemají nastaveny proměnné prostředí. Toto lze opravit buď nastavení těchto cest pomocí ve spouštěcím skriptu nebo pomocí C++ kódu.

Příklad 13: Nastavení umístění Qt Mobility pomocí proměnných prostředí

```
#!/bin/sh
export LD_LIBRARY_PATH=/opt/qtm12/lib
export QML_IMPORT_PATH=/opt/qtm12/imports
/opt/foo/bin/foo
```

Příklad 14: Nastavení umístění Qt Mobility podmíněným překladem

```
#if defined(Q_WS_MAEMO_5)
viewer->engine()->addImportPath(QLatin1String("/opt/qtm12/imports"));
viewer->engine()->addPluginPath(QLatin1String("/opt/qtm12/plugins"));
#endif
```

Pro správné zobrazení aplikace je třeba opravit nastavení orientace telefonu.

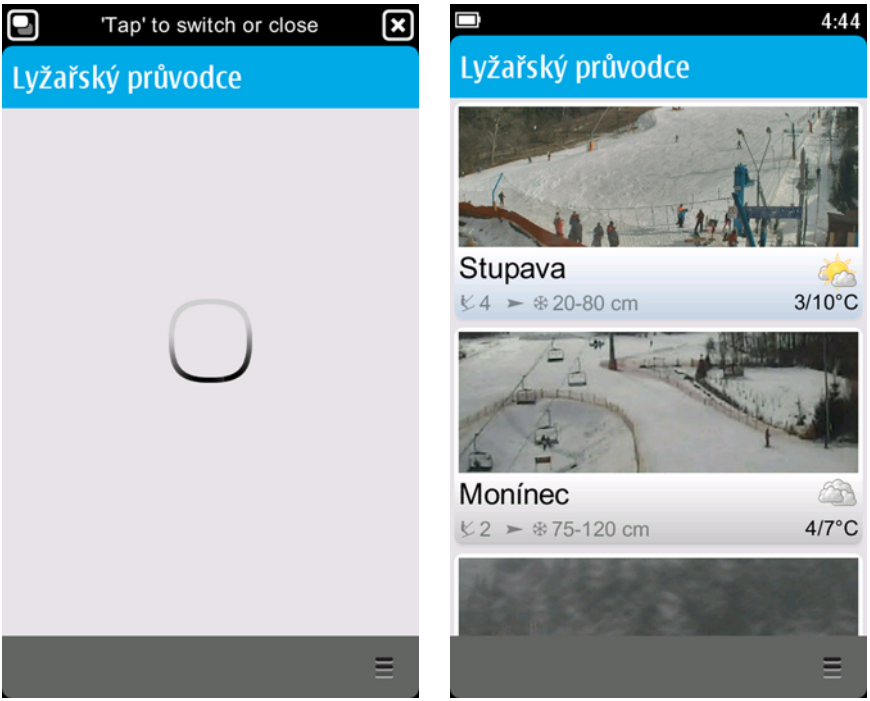
Příklad 15: Nastavení orientace zařízení

```
#if !defined(Q_WS_MAEMO_5)
viewer->setOrientation(QmlApplicationViewer::ScreenOrientationAuto);
#endif
```

V Maemu se používá standardní záhlaví pro aplikace s ikonami pro zobrazení seznamu aplikací, ukončením aplikace a systray. Použití tohoto standardního záhlaví by bylo komplikované a řeší se to tak, že komponenta `PageStackWindow` si vykresluje vlastní záhlaví aplikace . Aby se nezobrazovala původní záhlaví, tak je potřeba spustit aplikaci v celo obrazovém režimu:

Příklad 16: Potlačení standardního záhlaví aplikace

```
#if defined(Q_WS_MAEMO_5)
viewer->showFullScreen();
#else
viewer->showExpanded();
#endif
```



a) b)

Obr. 3: Záhlaví aplikace pomocí qt-components v systému Maemo

Obrázek 3 zobrazuje vzhled záhlaví. Při načítání aplikace se na okamžik objeví záhlaví s popiskem. Následně se zmenší, tak aby nezabíralo moc místa.

Pro kompatibilitu se systém *Maemo Fremantle* je potřeba opravit ještě další části zdrojového kódu aplikace. Jedná se například o `Pinch` {}, který bude ne-

fungovat na rezistivní obrazovce (bez podpory *multi touch*). Ve starší verzi je nutné kód zakomentovat, v novější to prostě jenom nebude fungovat. Dalších takových drobností, které bohužel nefungují, je celá řada. Tyto drobnosti bohužel nutí vývojáře založit další vývojovou větev kvůli kompatibilitě. Asi nejhorším problémem je vzájemná nekompatibilita jednotlivých verzí Qt Components v Maemu. Jinak řečeno, pokud má program fungovat za všech okolností, tak je potřebné vytvořit dvě verze QML souboru a podmíněným překladem vybírat, která z nich se má použít, přičemž lze CSSU verzi rozlišit jako verzi využívající Qt 4.7.4 a non-CSSU jako verzi s Qt 4.7.0.

Dalším krokem je vytvoření balíčku a jeho publikování. Pro vytvoření balíčku je možné použít tlačítko Publish v Qt Creatoru, bohužel ani tohle nefunguje správně. Pro publikování se v adresáři /tmp vytváří soubory potřebné pro odeslání balíčku do „build service“. Prvním problémem je nakopírování souboru debian/rules se špatnými právy, takže jej pak nelze spustit. Druhým problémem je, že soubor debian/rules sice obsahuje příkazy potřebné pro překlad programu, ale ty jsou zakomentované.

Příklad 17: workaround pro nastavení oprávnění

```
#!/bin/sh
while [ true ]; do
  chmod +x /tmp/qtc_packaging/projectname/debian/rules
done
```

Takhle připravený balíček je možné poslat do extras-devel pomocí extras asistenta nebo pomocí dput nebo přes webové rozhraní. Vytvořený balíček obsahuje zdrojový kód. Ten se na serveru přeloží a výsledný binární program se nakonec zveřejní v repositáři. Emailem příjde oznámeni o výsledku vzdáleného překladu.

Při vývoji programů využívající *Qt Components* v systému *Maemo Fremantle* je vhodné prostudovat strukturu dalších projektů, které využívají *Qt* a *Qt Components*. Jedná se například o projekty butaca, meecast nebo miniature.

5 Závěr

Tento článek popisuje omezení systému *MeeGo* pomocí jeho rozšíření *Aegis*. Dále se zabývá podobnými omezeními na systému *Symbian* a odlišnostmi vývoje aplikací s využitím frameworku *Qt* v tomto systému ve srovnání se systémem *MeeGo*. Na závěr je popsaná kompatibilita těchto systémů se systémem *Maemo Harmattan*.

Poděkování

Tato práce byla podporována projektem Získávání kontextu na mobilních zařízeních, FRVŠ MŠMT, FR1966/2012/G1.

Literatura

[1] *Global reach statistics.* http://www.developer.nokia.com/Distribute/Statistics.xhtml, 2012.

[2] *Qtcreator does not generate the required dependency for symbian qt quick components.* https://bugreports.qt-project.org/browse/QTSDK-1283, 2012.

[3] *Support for symbian, qt wiki, qt developer network.* http://qt-project.org/wiki/Support-for-Symbian, 2012.

[4] Fitzek, F. H. P., Mikkonen, T., Torp, T.: *Qt for Symbian.* John Wiley & Sons, 2010.

[5] Molkentin, D.: *The book of Qt 4: the art of building Qt applications.* No Starch Press Series. No Starch Press, 2007.

[6] Zucker, D., Rischpater, R.: *Beginning Nokia Apps Development.* Apress, Berkely, CA, USA, 1st edition, 2010.

Windows Phone 8 a Windows 8 – sdílení kódu

Štěpán Bechynský

E-mail: Stepan.Bechynsky@microsoft.com

V současné době existuje velké množství platforem pro vývoj aplikací od různých výrobců, které vyžadují více či méně rozdílný přístup ze strany tvůrců aplikací. Roztříštěnost různých platforem, zejména mobilních, pak vede k vytváření různých knihoven, jako je PhoneGap, které jsou většinou postaveny na HTML5 a snaží se rozdíly mezi platformami řešit. Výsledek pak hlavně záleží na schopnostech webového prohlížeče na konkrétní platformě.

S problémem roztříštěnosti se potýkají i platformy vytvářené jednou společností, společnost Microsoft nevyjímaje. Jak tento problém řeší poslední verze mobilní platformy Windows Phone 8 a Windows 8, resp. Windows Store, aplikace?

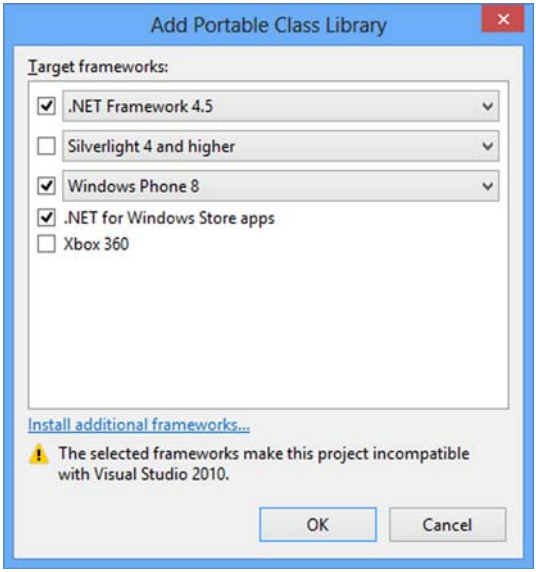
Portable Class Library

Portable Class Library je speciální druh knihovny, která je společná pro různé platformy.

Zjednodušeně řečeno, tento typ knihovny může obsahovat pouze kód, který je na 100 % identický na všech zvolených platformách. Obsahuje tedy průnik funkcionality vybraných platforem. Funkčnost, která je specifická pro konkrétní platformu, se pak izoluje do částí aplikace, které jsou již platformě závislé. Tento způsob vývoje aplikací je vhodný pro použití s návrhovým vzorem Model View ViewModel (MVVM) [http://en.wikipedia.org/wiki/Model_View_ViewModel]. Uživatelské rozhraní, View, je typicky platformě závislé.

Roztříštěnost mobilních platforem z hlediska správce IT

Často přehlíženými požadavky při nasazování mobilních platforem ve firmách, jsou požadavky správců IT. Tyto problémy se zejména týkají bezpečnosti. Ztráta zařízení pro společnost je většinou zanedbatelná položka v porovnání se ztrátou



informací, které jsou uloženy na mobilním zařízení. Velmi často je také požadována možnost vzdálené instalace aplikací na zařízení a celkové zařazení mobilního zařízení do firemní síťové infrastruktury.

Nová platforma Windows Phone 8 je z hlediska správců IT asi nejlépe kontrolovatelnou mobilní platformou. Je to dáno sdílením jádra operačního systému mezi Windows 8 a Windows Phone 8. Mobilní zařízení s operačním systémem Windows Phone 8 poskytují takřka stejné možnosti pro správu zařízení jako „velký“ Operační systém Windows 8. Z hlediska bezpečnosti, v závislosti na použitém hardware, je velmi zajímavá podpora Bitlocker, tedy šifrování dat na úrovni hardware. Díký sdílenému jádru operačního systému je možné aplikovat postupy správy operačního systému Windows 8 na zařízení s Windows Phone 8.

Závěr

Roztříštěnost platforem, nejen mobilních, je problém nejen pro vývojáře, ale i pro správce IT. Nové platformy společnosti Microsoft, Windows 8 a Windows Phone 8, se tuto situaci snaží řešit sdílením jádra operačních systému a tím pádem u možností sdílet velkou část kódu aplikace.

Android

Matěj Konečný

E-mail: shulik@shulik.cz

Android je v dnešní době nerozšířenějším a zároveň nejrychleji rostoucím operačním systémem. Po androidích programátorech je veliká poptávka, která rozhodně převyšuje nabídku. Pokud tedy nevíte, co dělat, Android by mohl být dobrou volbou.

Další výhodou Androidu je to, že se pro něj programuje v Javě. Znáte-li syntaxi Javy, máte půl hotovo. Teď jste se možná zhrozili, že budu povídat něco o MIDletech (tak se vyvíjely J2ME aplikace pro staré telefony). Nebojte. Android sice rozumí Javě (a máte-li vytvořenou nějakou knihovnu, půjde spustit i na Androidu), ale přidává k ní velké množství vlastních tříd, s pomocí nichž se aplikace vytváří. A tyto třídy jsou ta druhá polovina, těm jsem se věnoval na přednášce a těm se budu věnovat teď. Pokusím se vám velmi stručně shrnout všechny základní androidí koncepty.

Protože jsem limitován délkou tohoto článku, ke každému tématu se vyslovím jen velmi stručně a odkážu vás na příslušný díl mého seriálu Vyvíjíme pro Android, který mi v létě 2012 vycházel na Zdrojáku a který doporučuju pročíst, myslíte-li to s Androidem vážněji.

Instalace

Ale předtím, než si stručně představíme, jak pro Android programovat, si musíme nainstalovat všechny potřebné programy.

V první řadě potřebujete mít nainstalované JDK (opravdu JDK, jen JRE nestačí). Dokonce je potřeba JDK od Oraclu, s OpenJDK není Android (prý) kompatibilní.

Kromě JDK potřebujeme i nějaké IDE. Standardem je Eclipse[1], a to ve verzi Classic či Java. Pokud vám Eclipse nevyhovuje, jiným IDE, které podporuje Android, je IntelliJ IDEA. A samozřejmě lze (ačkoli to není dvakrát pohodlné) psát v Notepadu či jiném textovém editoru a všechno kompilovat, balit a instalovat na virtuální či reálné zařízení ručně.

[1] http:///www.eclipse.org/

Máme-li nainstalované JDK, můžeme stáhnout (a někdy nainstalovat) i ADK (Android SDK). To stáhnete z URL http://developer.android.com/sdk/index.html.

Eclipse nemá ve výchozím stavu podporu pro Android, tu musíme doinstalovat. K tomu slouží tzv. ADT (Android Development Tools) plugin, který stáhnete na stránce http://developer.android.com/sdk/installing/installing-adt.html. Díky ADT pluginu umí Eclipse pracovat s virtuálními či připojenými hardwarovými zařízeními, umí hotovou aplikaci zabalit a podepsat, umí napovídat v různých androidích XML souborech, přičemž pro většinu nabízí dokonce různé formuláře, jež vás od XML odstíní (ačkoliv já si myslím, že je to medvědí služba), a nabízí i WYSIWYG editor uživatelského rozhraní.

Než můžete začít programovat, ještě musíte spustit Android SDK Manager (v Eclipse Window Š SDK Manager) a v něm stáhnout všechny potřebné balíčky. Určitě doporučuju stáhnout nejnovější verzi SDK Platform a potom ty verze SDK, na nichž chcete aplikaci testovat. Kromě toho určitě nainstalujte celou složku Tools a ze složky Extras ještě Android Support, a jste-li na Windows, také Google USB Driver a Intel Hardware Accelerated Execution Manager.

Rady, jak rozchodit virtuální či hardwarová zařízení, stejně tak jako podrobnější návod k instalaci najdete ve Vyvíjíme pro Android: Začínáme.

Jen bych vás ještě chtěl upozornit na jednu věc. Android se velmi rychle vyvíjí (v podstatě co půl roku, to nová vere, která obsahuje zásadnější novinky) a s ním se rychle vyvíjejí i pomocné nástroje (například zmíněný ADT plugin). Proto to, co čtete v tomto sborníku už ve chvíli, kdy to čtete, nemusí být pravda. Neznamená to, že by vaše aplikace přestala fungovat – Android je zpětně kompatibilní – ale mohou se změnit nástroje, z *best practice* se může stát *deprecated* postup, mohou přibýt nové možnosti, ale také povinnosti, které musíte splnit, aby vaše aplikace na nových verzích Androidu vypadala moderně.

Pokud se s nějakou takovou nekonzistencí setkáte, doporučuju začít hledání aktuálnějších informací na Zdrojáku (můj autorský profil se seznamem všech článku je na adrese http://www.zdrojak.cz/autori/matej-konecny/). Možná jsem tam už stihl informace aktualizovat.

Založení nového projektu

Chceme-li vytvořit novou aplikaci pro Android, musíme pro ni v Eclipse vytvořit projekt. Ten bude seskupovat všechny potřebné soubory, různou konfiguraci pro Eclipse a tak podobně. Díky ADT pluginu se nám navíc při vytvoření nového projektu jednak vytvoří adresářová struktura a několik potřebných souborů a tříd, jednak v něm můžeme některé věci nastavit, a dokonce nám nabídne několik šablon. Průvodce je poměrně intuitivní, aktualizovaná verze návodu, jak založit nový projekt, je dostupná na URL http://www.zdrojak.cz/clanky/vyvijime-pro-android-content-providery/#new-project (pro ADT plugin v20).

Jak vypadá taková androidí aplikace?

Pomineme services, broadcast receivery, widgety, content providery a podobné a budeme se soustředit čistě na základní uživatelské rozhraní aplikace. Pak můžeme říct, že obvykle platí „co obrazovka, to Activity“. `Activity` je tedy základním stavebním kamenem uživatelského rozhraní. Můžeme si ji představit jako jednu statickou webovou stránku. Stejně tak jako z hlavní stránky nějakého e-shopu můžeme přejít na seznam produktů, nákupní košík nebo kontaktní formulář, což jsou všechno další webovou stránky, jedna Activity může spouštět jiné Activity, například detail nějakého e-mailu, formulář pro vytvoření nové zprávy či nastavení. Dokonce, stejně tak jako jedna webovou stránka může odkázat na jinou na úplně cizí doméně, může jedna Activity spustit jinou, která přísluší úplně cizí aplikaci (na Androidu je to ještě chytřejší, budeme si o tom povídat později).

„No jo,“ řeknete si, „ale webová stránka se skládá z HTML, které definuje její strukturu, CSS, které definuje její vzhled, a Javascriptu, který zajišťuje padající vločky, vyskakovací okna a monstrózní obludná vyjíždějící menu s cool efektem. Android hadr!“ Není to pravda. Activity a webová stránka jsou si podobnější, než si myslíte.

View

Activity slouží jako kontejner pro různá View. To znamená, že musíme definovat, že například v `MailFormActivity` budeme mít dvě jednořádková vstupní textová políčka (pro příjemce a předmět), jedno víceřádkové (pro tělo e-mailu) a tlačítko pro odeslání, a k tomu slouží právě View. View `EditText` poslouží pro všechna tři textová políčka a View `Button` ztělesňuje tlačítko. A aby byla analogie ještě přesnější, je možné (a doporučené) definovat View v XML souboru.

Styly

U webové stránky můžeme pro stylování použít CSS. Android také nabízí styly (nutno dodat, že ale žádné kaskády neumí), pomocí nichž můžeme přesný vzhled oddělit od struktury a zároveň ho použít vícekrát.

Activity

Samotná Activity pak zastupuje Javascript. S Javascriptem má společnou i orientaci na události – jednak má Activity určité metody životního cyklu, které se spouští podle toho, v jakém stavu se Activity nachází (když se Activity vytváří, spustí se metoda `onCreate()`, když je překryta jinou Activity, zavolá se `onStop()` atp.), jednak se může dobrovolně přihlásit k dalším událostem, ať už

událostem svých View (kliknutí na tlačítko, změna textu v `EditText`-u, ...) či „activitovým“ událostem (klepnutí na tlačítko zpět, napsání něčeho na hardwarové klávesnici, přestože není aktivní žádné View, které by se o to postaralo, a.j.).

Manifest

Manifest (soubor `AndroidManifest.xml`) je speciální soubor, který mimojiné deklaruje všechny části vaší aplikace. Pro nás to znamená, že tam musí být zapsány všechny Activity. Activity vytvořená při vytváření nového projektu už v manifestu zapsaná je, jak zapsat další se dozvíte ve článku Vyvíjíme pro Android: První krůčky.

1 Jak tedy může vypadat kód takové hodně jednoduché aplikace?

Naše aplikace se bude skládat z jedné Activity, která bude tudíž mít jeden XML soubor s jedním View – tlačítkem. Po klepnutí na to tlačítko se zobrazí nějaká zpráva.

Vynechal jsem deklaraci namespace a importy (ty za vás, mimochodem, vyřeší Eclipse stisknete-li `Ctrl + Shift + O`).

Javový soubor nazveme `MainActivity.java` a uložíme ho do `src/namespace naší aplikace`.

```
public class MainActivity extends Activity {

    // Zavolá se při vytvoření nové Activity
    @Override
    public void onCreate(Bundle savedInstanceState) {
        // Vždy musíme zavolat rodičovskou metodu onCreate()
        super.onCreate(savedInstanceState);

        // Tímto nastavíme, že se v˜Activity zobrazí naše tlačítko
        setContentView(R.layout.activity_main);

        // Získáme tlačítko pomocí metody findViewById()
        Button button = (Button)findViewById(R.id.button);

        // Přidáme tlačítku posluchače kliknutí
        button.setx_onClickListener(new x_onClickListener() {
```

```
                        public void x_onClick(View v) {
                                showMessage();
                        }
                });
    }

    private void showMessage(){
        // Toast.makeText() zobrazí takový ten malý černý
        obdelníček se zprávou
        Toast.makeText(this, "Byla provedena realizace kliknutí
        na tlačítko.", Toast.LENGTH_SHORT).show();
    }
}
```

XML soubor s tlačítkem nazveme `activity_main.xml` a uložíme ho do `res/layout`. Bude vypadat následovně:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/
       res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="Klikni na mě!" />

</RelativeLayout>
```

Vysvětlování jednotlivých metod a elementů je nad kapacitu tohoto příspěvku, takže vás odkážu na své články na Zdrojáku, kde jsem to vysvětlil velmi podrobně. O struktuře aplikace, co to je Activity, View atd. jsem psal ve článku Vyvíjíme pro Android: První krůčky. Různým View jsem se věnoval ve dvojdílu Vyvíjíme pro Android: Bližší pohled na pohledy – 1. díl potažmo Vyvíjíme pro Android: Bližší pohled na pohledy – 2. díl. Životní cyklus (nejen) Activit jsem představil ve článku Vyvíjíme pro Android: Dialogy a activity.

Přesto se tu ale (stručně) zmíním o dvou věcech.

Layouty

Třída `View` má podtřídu `ViewGroup`, která se od ní liší tím, že může obsahovat další View (a tedy i ViewGroup). Hlavními představiteli `ViewGroup` jsou třídy končící na Layout. Jde například o použitý `RelativeLayout`, ale běžně se používá i `LinearLayout` či `FrameLayout`. Tyto Layouty umožňují nějakým způsobem rozmístit View v nich obsažené.

Resources

Místo slova `resources`, které nemohu skloňovat, budu používat slovo `suroviny`.

Suroviny jsou způsob, jak v javovém kódu přistupovat k různým souborům ze složky `res`. A nejde jen o obyčejné čtení jakýchkoli souborů (ty mimochodem patří do podsložky `raw`), v případě speciálních XML souborů můžete například definovat *layout* (s malým l to znamená strom View, který se pak zobrazí v nějaké Activity), a to v podsložce `layout`, či menu (`menu`). Do surovin patří také obrázky (`drawable`). Speciální podsložku pak tvoří `values`, do nichž spadá mnoho různých typů XML souborů. Nejčastěji používaným je soubor s řetězci, které se používají v uživatelském rozhraní aplikace.

Podsložky složky res podporují také tzv. *modifikátory*. Je to speciální řetězec, který se přidá za pomlčku následující název složky a který určuje nějaké podmínky, při nichž se použije právě obsah dané složky. Asi nejlépe si to ukážeme na příkladě s překlady. Řetězce se obvykle dávají do souboru `res/values/strings.xml`. Pokud chcete mít českou, slovenskou a anglickou mutaci aplikace, bez znalosti modifikátorů by to bylo docela složité. S nimi je to v podstatě jen vytvoření souboru `res/values-cs/strings.xml` s českými řetězci, `res/values-sk/strings.xml` se slovenskými řetězci a `res/values/strings.xml` (které se použije, pokud neexistuje žádná specifičtější složka) s anglickými řetězci. Která data se použijí, se rozhodne sám Android za běhu na základě toho, jaký jazyk má uživatel nastavený.

A jak k surovinám přistupovat?

Na to existuje automaticky generovaná třída `R`. Ta má různé podtřídy (`R.drawable, R.menu, R.layout, R.string`, ...), které obsahují unikátní identifikátory daných surovin. Máme-li například v našem příkladu layoutový soubor nazvaný `activity_main.xml`, jeho identifikátor je v `R.layout.activity_main`. Máme-li řetězec s id `title`, jeho identifikátor je `R.string.title`, který zastupuje vybraný řetězec na základě aktivního jazyka. Jen vás upozorním, že `R.string.asd` či cokoli jiného obsahuje jen identifikátor. Chcete-li získat přímo konkrétní řetězec, je to trochu složitější.

@+id/button... ehm?

V XML se pro přístup k surovinám používá notace `@string/title`, `@layout/activity_main` apod. Existuje také třída `R.id`, která sdružuje *identifikátory jednotlivých identifikátorů*. A protože id přidáváme prakticky jen v layoutu, Android nám poskytuje zjednodušení právě v podobě `@+id`, které vytvoří nový identifikátor s daným jménem (zde `button`) a přiřadí ho tomu View. Úplně stejný efekt by mělo přidat do `res/values/ids.xml` id se jménem `button` a do atributu `android:id` napsat jen `@id/button`.

Jsem si vědom toho, že zdejší opravdu stručný úvod nemusí být příliš jasný. Úvod do surovin a identifikátorů je ve článku Vyvíjíme pro Android: První krůčky, detailně je pak probíráme ve Vyvíjíme pro Android: Suroviny, Intenty, jednotky (kde se navíc věnujeme i rozměrovým jednotkám a Intentům, o nichž budu povídat teď).

Intent

`Intent` je velmi primitivní, ale přitom mocná třída, která v Androidu zajišťuje (nejen) veškeré spouštění Activity.

Překlad slova *intent*, tedy *záměr*, poměrně přesně vystihuje, co objekt Intent znamená – nějaký záměr, nějaký úkol, který potřebujeme splnit a androidí framework nám s tím má pomoci.

Základní využití Intentů je pro spouštění Activit. Třída `Activity` má (mimojiné) metodu `startActivity(Intent i)`, která přebírá právě objekt `Intent`. A jak má takový Intent vypadat?

Explicitní Intent

Nejjednodušším a minimálně ze začátku rozhodně nejpoužívanějším bude takzvaný *explicitní Intent*. Je to Intent, který přímo specifikuje třídu Activity, která se má spustit. Vypadá třeba takto:

```
Intent explicitIntent = new Intent(context, MyCoolActivity.class);
```

přičemž `context` je instance třídy `Context`. Jedním z potomků `Context`-u je i třída `Activity`.

Implicitní Intent

Implicitní Intent je mnohem zajímavější, neboť umožňuje specifikovat, jakou akci chcete provést, místo toho, kdo ji má provést. Příslušnou Activity pak vybere Android, případně nechá zvolit uživatele, je-li možných cílů více.

Implicitní Intenty nejsou sice nijak složité na pochopení, ale vzhledem k tomu, že toho docela dost umí, odkážu vás na článek Vyvíjíme pro Android: Intenty, intent filtry a permissions, kde se kromě Intentů věnuji i intent filtrům, což je způsob, jak dát vědět, že vaše Activity dokáže splnit určitý implicitní Intent.

Fragmenty

Android 3.0 Honeycomb byl prvním Androidem optimalizovaným pro tablety. Kromě změny uživatelského rozhraní samotného Androidu musel ovšem přijít i s mechanismem, jak přizpůsobit aplikace. A tím mechanismem jsou takzvané Fragmenty.

Takový Fragment si můžeme velmi zjednodušeně představit jako trochu osekanou Activity. Osekanou v tom smyslu, že nemůže žít sám o sobě, ale musí být „obsažen“ v nějaké Activity. Jinak sám může obsahovat View (bohužel nikoli další Fragmenty) a má také metody životního cyklu.

Pravá síla Fragmentů se ovšem objeví až tehdy, kdy je použijeme spolu s modifikátory surovin. Existuje totiž i modifikátor (`wNdp`, kde `N` je číslo), který definuje minimální šířku displeje. A pokud s takovým modifikátorem vytvoříme např. složku `layout-w600dp` a do ní dáme layoutový XML soubor s více sloupci, do nichž umístíme Fragmenty (zjednodušeně řečeno), přičemž stále máme složku `layout` a v ní layoutový soubor (stejně pojmenovaný!) s jedním sloupcem, máme aplikaci přizpůsobenou pro tablety.

Abychom však mohli používat Fragmenty i na starších zařízeních a nemuseli tedy vytvářet dvě verze aplikace, musíme použít Fragmenty z tzv. *Support library*.

O Fragmentech i Support library jsem psal více ve článku Dej Androidu tablety! a Vyvíjíme pro Android: Fragmenty a SQLite databáze.

Ukládání dat

Poslední, čemu se budu věnovat, je krátký úvod do ukládání dat na Androidu.

V podstatě existují tři způsoby: uložit data jako soubor, použít `key-value` úložiště a SQLite databáze.

Souborové úložiště

Chcete-li ukládat a číst soubory, máte dvě možnosti. Interní úložiště dané aplikace, k němuž nemá nikdo jiný přístup a jež je smazáno současně s aplikací, anebo externí úložiště, které je přístupné všem a soubory tam zůstanou i po odstranění aplikace ze zařízení. Práce se souborovým úložištěm je (stejně tak

jako ostatní možnosti ukládání dat) prodiskutována v dokumentaci na URL http://developer.android.com/guide/topics/data/data-storage.html.

Key-value úložiště

Key-value úložiště se na Androidu jmenuje `SharedPreferences` a už název napovídá, že se používá hlavně pro ukládání uživatelského nastavení. To vám ale nebrání použít ho jakkoli chcete, ačkoli to není vždy tak příjemné jako SQLite databáze `SharedPreferences` jsem se věnoval ve článku Vyvíjíme pro Android: Preference, menu a vlastní Adapter.

SQLite databáze

Poslední a asi nejpoužívanější možností ukládání dat je vestavěná SQLite databáze. Je to plnohodnotná SQLite databáze, s níž můžete komunikovat pomocí SQL, ačkoli pro vkládání, mazání, upravování a získávání dat se používá jiné API, které však samozřejmě samo SQL využívá. Je to sice rozhodně zajímavé téma, ale vystačilo by na samostatnou přednášku a samostatný příspěvek do sborníku a kromě toho jsem se mu poměrně svědomitě už věnoval.

Odkazy

Kromě už zmíněného seriálu Vyvíjíme pro Android vám můžu doporučit stránky Android Developers, kde najdete dokumentaci, API referenci i design guidelines. Příliš dalších dobrých zdrojů neznám, snad ještě stránky Larse Vogelly.

Mě potom můžete sledovat na Twitteru (@MatejKonecny), a pokud máte dotaz nebo něco jiného na srdci, můžete mi napsat e-mail (shulik@shulik.cz).