VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Komprese XML souborů

Compression of XML Files

2010 Pavel Hruška

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava*.

V Ostravě 7. května 2010 .....

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2010 .....

Děkuji všem, kteří mi pomáhali během příprav této diplomové práce, především vedoucímu práce Ing.Janu Martinovičovi, Ph.D. za jeho ochotu, trpělivost a cenné rady.

Abstrakt

Práce s XML soubory je dnes čím dál tím více častější. Existují také XML dokumenty, které obsahují velké množství dat. Tato diplomová práce popisuje existující algoritmy používané ke kompresi XML dokumentů a také popisuje některé nové způsoby, jak stávající přístupy vylepšit. Zaměřuje se na několik populárních kompresních algoritmů a jejich použití jak při kompresi XML jako textu, tak i při kompresi XML s využitím sémantických informací dostupných v XML dokumentech. Dále popisuje rozšíření těchto metod o optimalizaci XML pomocí shlukování. Na základě provedených testů jsou porovnány efektivnosti jednotlivých algoritmů a vysloven závěr, zda lze rozšířením stávajících metod komprese XML dokumentů dosáhnout lepších výsledků komprese.

Klíčová slova: komprese, komprese textu, XML, shlukování dokumentů, analýza XML.

Abstract

Working with XML files is now becoming more frequent. There are XML documents containing large amount of data. This thesis deals with existing algorithms used for XML compression and some new ways of improving current approaches. This thesis focuses on some popular text compression algorithms and their application either in standard text file compression or in XML compression through semantic information that is present in XML documents. The thesis also describes extending the methods with XML optimization through agglomerative clustering. Various compression methods are compared on the basis of testing in order to find out whether XML compression methods extension can achieve better results.

Keywords: Compression, Text Compression, XML, Documents Clustering, Parsing XML.

Obsah

Seznam tabulek

Seznam obrázků

1 Úvod

XML [29] je v dnešní době velmi rozšířený jazyk pro ukládání a výměnu dat. Přes své nesporné výhody, které pramení především z jeho univerzálnosti a také z textové podoby jeho datového formátu (dobře čitelného pro člověka), má i některé nevýhody. Mezi hlavní patří především nutnost analýzy XML dat před jejich použitím. Pokud jej srovnáme s nativními binárními formáty, jedná se také o relativně výřečný jazyk, který klade větší nároky na prostor nutný k uložení reprezentovaných dat. V této práci stručně charakterizujeme XML a zabýváme se také problematikou spojenou s jeho analýzou.

Problém výřečnosti XML lze řešit pomocí komprese dat. Jelikož se na XML dá dívat z více pohledů a i na samotnou kompresi XML se mohou klást různé požadavky, je možné také k problematice komprese XML přistupovat více způsoby. Jelikož je XML v podstatě textový dokument, je přirozená myšlenka komprimovat jej *běžnými kompresními nástroji*, které denně používáme pro kompresi jiných nejen textových dokumentů a obecně souborů. Tyto nástroje využívají osvědčené kompresní algoritmy, které jsou díky bohaté historii prakticky ověřené a velmi často využívané jako spolehlivé prostředky ke kompresi dat. Jejich použití je tedy velmi snadné a výsledky se dostavují okamžitě. V této práci testujeme kompresi XML souborů pomocí algoritmů Deflate, BZip2, PPMdI a LZMA.

Kompresi XML lze řešit i specializovanými nástroji, ty se nazývají *XML-aware kompresory*. V této práci uvádíme přehled nástrojů a algoritmů, které jsme shromáždili studiem odborných článků a internetových zdrojů věnujících se XML kompresi.

Dále se podrobněji věnujeme nástroji *XMill*. Popisujeme zde více detailněji jeho princip komprese, datový formát a další informace. S tímto nástrojem pak dále pracujeme, především s naší vlastní implementací v prostředí .NET Framework (v jazyce C#). XMill jsme si zvolili především proto, že se jedná o velmi populárního zástupce XML-aware komprese, který velmi často slouží jako etalon při srovnání odlišných přístupů komprese XML. Navíc je velmi dobře zdokumentován.

Po detailním popisu XMill dále popisujeme naší implementaci tohoto nástroje, kterým je *SXMill* (SharpXMill). Zaměřujeme se především na základní architekturu navrženého systému a na jeho odlišnosti oproti původnímu XMill.

Dále se věnujeme problematice shlukování dat, především využití této metody ve spojení s kompresí XML. Konkrétně jsme shlukování dat využili k optimalizaci XML a experimentovali jsme, zda nepovede k dosažení lepších výsledků komprese. Vyzkoušeli jsme dvě možné cesty optimalizace — *shlukování dat v kontejnerech* během XMill komprese a *shlukování celých XML souborů*.

V závěru práce pak vyhodnocujeme provedené experimenty. Testovali jsme běžné metody komprese dat i navržené optimalizace. Veškeré výsledky srovnáváme v kontextu jednotlivých metod komprese a nástrojů.

1.1 Struktura práce

Charakteristikou XML a jeho výhodami a nevýhodami se zabýváme v sekci 2. V sekci 2.4 se věnujeme problematice zpracování XML a nejčastěji používaným modelům zpra-

cování XML — XML DOM a SAX. Kompresi XML je věnována sekce 3, konkrétně v sekci 3.2 popisujeme kompresi XML jako textu (včetně popisu jednotlivých metod komprese) a v sekcích 3.3, 3.4 a 3.5 se zaměřujeme na kompresi specializovanými (XML-aware) nástroji, kde postupně rozebíráme kompresi XML s podporou dotazovaní a následně bez podpory dotazování. Sekce 4 se podrobně věnuje nástroji XMill, který je typickým představitelem kategorie XML-aware komprese bez podpory dotazování. V sekci 4.1 popisujeme jeho architekturu, v sekci 4.2 uvádíme na jednoduchém příkladu jím používaný princip oddělení struktury od dat a v sekci 4.3 popisujeme datový formát tohoto nástroje. Sekce 5 popisuje námi implementovaný nástroj SXMill (SharpXMill). Jeho architektura je popsána v sekci 5.1, podporované kompresní metody pak v sekci 5.2 a popis rozšíření oproti původnímu XMill je uveden v sekci 5.3.

Sekce 6 se věnuje prezentaci výsledků provedených experimentů. V sekci 6.2.1 jsou k dispozici výsledky komprese XML souborů jako textu (běžná komprese), v sekci 6.2.2 uvádíme výsledky komprese pomocí nástroje XMill. Sekce 6.2.3 se zabývá otázkou, jak u nástroje XMill ovlivňuje velikost paměťového okna jeho úspěšnost komprese při použití jednotlivých kompresních metod. Sekce 6.2.4 se věnuje kompresi XML pomocí nástroje XMill s optimalizací kontejnerů využívající shlukování. V sekci 6.2.5 ukazujeme výsledky komprese XML běžnou kompresí po provedení shlukování celých XML souborů.

V sekci 7 shrnujeme výsledky experimentů a vyslovujeme závěr, zda je možné pomocí shlukování vylepšit stávající metody komprese XML souborů.

2 XML

XML (eXtensible Markup Language, česky rozšiřitelný značkovací jazyk) je obecný, otevřený značkovací jazyk, standardizovaný konsorciem W3C. Vznik tohoto jazyka se datuje do roku 1998, kdy byla standardizována verze 1.0. XML je založen na obecném metajazyku SGML, přesněji řečeno tvoří jeho podmnožinu. Ve srovnání s SGML je jednodušší, snadněji se analyzuje [1] [29, 32].

2.1 Charakteristika XML

XML je obecný jazyk, který umožňuje definovat vlastní jazyky — představuje sadu pravidel, které se používají k definici konkrétních jazyků. V dnešní době se využívá především jako prostředek pro výměnu dat v prostředí internetu, např. u řešení B2B[2] [5] apod. Díky své univerzálnosti mu dávají vývojáři přednost u systémů, u kterých není v době návrhu předem jasné, s jakými dalšími systémy bude nutné komunikovat. XML nachází uplatnění také jako univerzální datové úložiště, velmi populární jsou v dnešní době také na XML založené konfigurační soubory.

XML dokument je tvořen posloupností znaků, kde znakem se rozumí libovolný *Unicode* znak. XML je tedy ve své podstatě textový dokument. V XML dokumentu se rozlišují dva základní elementy — *značky* a *obsah*. Značky jsou uvozeny znakem „<“ (menší) a končí „>“ (větší), nebo začínají znakem „&“ (ampersand) a končí znakem „;“ (středník). Vše ostatní, co není značka, je obsah [29].

XML je strukturovaný jazyk, jeho struktura se tvoří vzájemným vnořováním značek. Jako každý jazyk, má i XML svou syntaxi. Každý XML soubor musí splňovat minimálně pravidla *well-formed XML*[3] zápisu. Well-formed XML nedovoluje například použití překřížených značek [29].

Na jazyku XML jsou postaveny některé další jazyky. Jedná se například o RSS [34] a Atom [20] (syndikace obsahu), SOAP [36] (výměna zpráv), XML-RPC (vzdálené volání procedur) [40] či XHTML [37] (rozšiřitelný hypertextový značkovací jazyk). Mnoho aplikací začíná využívat XML také jako základní datový formát, například jako populární kancelářské balíky Microsoft Office (formát Open XML) [31] a OpenOffice.org (formát OpenDocument) [3].

2.2 Výhody XML

Jak již bylo řečeno, XML je velmi rozšířeným a hojně používaným jazykem v praxi. Jeho velké rozšíření pramení z nesporných výhod, kterými tento jazyk disponuje. Především se jedná o univerzální jazyk, do kterého je možné serializovat[4] a zpětně z něj deserializovat[5] libovolná data. Díky tomu se velmi často používá jako prostředník u vzájemné ko-

[1] Z anglického *parse*.
[2] Business-to-Business
[3] Syntaktická pravidla pro zápis XML dokumentů.
[4] Vytvoření proudu symbolů, které reprezentují stav nějaké informace.
[5] Reverzní operace k operaci serializace.

munikace mezi různými systémy, které vnitřně využívají odlišné formáty, ale pro výměnu dat používají univerzální XML. Tím odpadá nutnost vytvářet specifické převodní můstky pro každý nový partnerský systém, se kterým je potřeba komunikovat, při jejichž tvorbě se dopodrobna musíme seznámit s formátem druhé strany. Namísto toho se definuje pouze XML rozhraní, se kterým pak může komunikovat jakýkoliv systém, aniž by znal vnitřní strukturu dat daného systému. Textový formát je v neposlední řadě také dobře čitelný pro člověka.

2.3 Nevýhody XML

Zmíněné výhody XML přinášejí na druhou stranu i jeho nevýhody. Mezi hlavní nevýhodu XML patří především jeho výřečnost. Výřečnost vede v porovnání s konkrétními datovými formáty k mnohem větší datové náročnosti XML dokumentů. Pokud s XML chceme pracovat, je také nutné nejdříve jej analyzovat (anglicky *parse XML*), což může především při zpracování rozsáhlejších XML znamenat významnou zátěž pro výpočetní výkon systému [19].

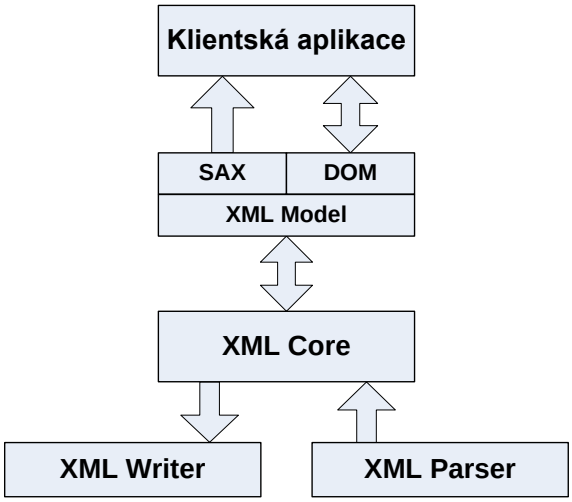
Tyto nevýhody jsou obecně známé, proto vznikl například *XML Binary*. Jedná se o standardizovaný formát, který se neshoduje se specifikací XML, ale pouze si zachovává jistý vztah s původním XML [39]. XML Binary tak lze použít u aplikací, u kterých může být výřečnost běžného XML problém, ale existuje u nich požadavek na využití standardizovaného formátu výměny dat.

2.4 Zpracování XML

XML data jsou představována sérií *Unicode* symbolů, datovým úložištěm jsou pak typicky textové soubory. Při práci proto musí běžné aplikace tuto lineární strukturu zpracovat — analyzovat — a identifikovat v ní jednotlivé prvky struktury a samotných dat. Samozřejmě je nutné mít k dispozici i inverzní operaci, tedy vytvoření a úpravu well-formed XML reprezentujícího danou strukturu a obsahujícího požadovaná data.

Existuje několik modelů zpracování XML, dva nejčastěji používané *SAX (Simple Api for XML)* [35] a *XML DOM (XML Document Object Model)* [28] stručně představíme v další části textu. U zpracování XML je velmi důležitou částí *analýza XML*, protože právě ona může mít zásadní vliv na chování cílové aplikace. Problematice analýzy XML se věnujeme v následujícím textu této kapitoly.

Obecný model zpracování XML dat popisuje způsob čtení a zápisu XML dat ve vztahu ke klientské aplikaci. Jeden z modelů je znázorněn na obrázku 1. Obrázek ukazuje na nejvyšší úrovni klientskou aplikaci, která čte, popř. mění XML data. Nižší vrstvy pak odstiňují tuto aplikaci od operací spojených s vlastním zpracováním XML dat. Díky tomuto modelu může aplikace přistupovat k XML na vyšší, abstraktnější úrovni. Pod klientskou aplikací jsou v modelu znázorněny vrstvy *XML Core* (ta obsahuje funkce pro zpracování XML dat dle konkrétních požadavků implementace či prostředí) a dvojice komponent *XML Writer* a *XML Parser* (ty slouží ke čtení a zápisu XML).



Obrázek 1: Obecný model zpracování XML

2.4.1 XML DOM (XML Document Object Model)

XML DOM (XML Document Object Model) vychází z obecné definice DOM (Document Object Model), což je jazykově a platformě neutrální rozhraní, které umožňuje programům a skriptům dynamicky přistupovat a aktualizovat obsah, strukturu a styl dokumentů [28].

XML DOM nahlíží na XML jako na strom, který se skládá z jednotlivých uzlů. Kořenovým uzlem je uzel na nejvyšší úrovni a ten může obsahovat uzly podřízené. Rekurzivně tato vlastnost platí i pro podřízené uzly, tedy podřízené prvky mohou obsahovat sobě podřízené prvky a tak dále.

Během analýzy vstupních dat se v paměti postupně vytvoří jim odpovídající strom. Úpravy pak probíhají v paměti a celý strom nebo jeho část je kdykoliv možné zapsat ve formátu XML. Z uvedeného způsobu práce je patrné, že XML DOM potřebuje mít ke své práci v paměti neustále celou strukturu i samotná data, což může činit problém při zpracování rozsáhlých XML dokumentů. Výhodou tohoto modelu je fakt, že aplikace může libovolně číst i měnit strukturu a data, protože DOM podporuje operace čtení i modifikace a to typicky objektově.

2.4.2 SAX (Simple API for XML)

SAX provádí postupnou analýzou vstupních XML dat a během ní identifikuje jednotlivé části XML dokumentu, jako jsou značky, atributy, entity, komentáře atp. SAX je založen na událostech, na které se klientská aplikace váže a pomocí nichž pak získává data. Hlavní rozdíl oproti dříve zmíněnému modelu DOM je v tom, že SAX neudržuje v paměti strukturu ani data celého dokumentu, ale pouze data aktuálně analyzované části [35]. Díky tomu je možné zpracovat libovolně rozsáhlý XML dokument, nicméně, z principu je možné data pouze číst a to pouze lineárně. V praxi se běžně SAX model implemen-

```
<položka>
  <nadpis>Formátování textu</nadpis>
  <popis>V textu můžeme použít <b>tučné písmo</b> ale ne <i>kurzívu</u>.</popis>
  ...
</položka>
<položka>
  <nadpis>Příklad platného záznamu</nadpis>
  <popis><b>Upozornění:</b> Nepoužívat!</popis>
  ...
</položka>
```

Obrázek 2: Příklad XML s nejasně analyzovatelnou strukturou

tuje jako *SAX parser*, což je konkrétní analyzátor podporující model SAX. SAX je vhodný při použití s aplikacemi, které potřebují vstupní data pouze číst a nevyžadují v jednom okamžiku komplexní pohled na celý XML dokument.

2.5 Analýza XML

XML se skládá ze značek a obsahu [29]. Značky jsou definovány pomocí speciálních znaků („<“ a „>“). Vše, co není značka, je považováno za obsah. Ne vždy ovšem může být rozdělení značek a obsahu zcela zřejmé. U well-formed XML dokumentu není problém identifikovat veškeré značky, nicméně některé značky nemusí mít význam značky jako *definující strukturu*, ale může se jednat o značku, která je umístěna v kontextu nějakého obsahu jako *formátovací* nebo jiný *pomocný prvek* (např. označení tučného textu značkou). To může mít za následek nestandardní chování cílové aplikace. Příklad konkrétního XML, který odpovídá podobnému popisu, je znázorněn na obrázku 2.

Způsob analýzy takového XML souboru má vliv na strukturu a obsah informací, jaké aplikace získá od analyzátoru. Dvě konkrétní úskalí popisují následující dvě podkapitoly.

2.5.1 Analýza struktury a obsahu

Ukázkový příklad obsahuje dle formální definice XML celkem čtyři jedinečné značky *položka*, *nadpis*, *b*, *i* a k nim odpovídající koncové značky. Nicméně tyto značky v daném kontextu spadají do dvou kategorií. První kategorii tvoří značky, pomocí kterých se tvoří požadovaná struktura dat — to jsou značky *položka* a *nadpis*. Vše ostatní, tedy texty umístěné uvnitř těchto značek, má být považováno za obsah. Ovšem vložení textu podobnému našemu příkladu tuto myšlenku rozbíjí. Značky *b* a *i*, které slouží v kontextu pouze jako formátovací prvky, vytváří v původní definici XML dodatečnou strukturu.

Jak je patrné, v tomto příkladu nejsou značky *b* a *i* značkami ve smyslu struktury XML, ale jsou součástí obsahu značky *popis*. A právě to může mít nežádoucí vliv na chování aplikace, pokud analyzátor nebude o definované struktuře dostatečně informován (např. XML schématem). Navíc, dané XML je v tuto chvíli velmi citlivé na zpracování formátování, neboli white-spaces.

2.5.2 Analýza formátování dokumentu (white-spaces)

S předstihem můžeme prozradit, že XMill[6] ve výchozím nastavení využívá optimalizaci komprese XML tím, že odstraňuje formátování XML (white-spaces), které pak vytváří při dekompresi programově. Další podrobnosti o nástroji XMill jsou v sekci 4.

XML podporuje celkem čtyři druhy white-spaces[7]: *carriage-return* (`\r`), *line-feed* (`\n`), *tab* (`\t`) a *spacebar* (`mezera`) [29]. Na první pohled se může zdát, že tyto informace nenesou obsahovou informaci, proto právě zmiňovaný XMill ve výchozím nastavení white-spaces ignoruje a během dekomprese je rekonstruuje programově [38].

Pokud se ale podíváme na námi uváděný příklad, bude mít použití této optimalizace vliv na obsah dat, protože při provádění dekomprese nebudou white-spaces rekonstruována korektně. V příkladu se za koncovou značkou `</b>` (mezi slovy *písmo* a *ale*) nachází mezera a při aktivní optimalizaci by se během komprese jako white-space symbol ignorovala. Tím by došlo ke ztrátě informace, protože při programové rekonstrukci formátování XMill nikdy nedává za značku mezeru, maximálně odsazuje následující obsah na další řádek.

Během našich experimentů jsme proto vždy tento druh optimalizace potlačili a to i u souborů, u kterých to vzhledem k jejich obsahu nebylo nutné. Toto opatření nám také zajistilo, že se po dekompresi soubory zcela shodovaly s původními originály. Jedinou nevýhodou je to, že se spolu s komprimovanými daty musí ukládat i vlastní formátování, což mírně zhoršuje výsledek komprese.

2.5.3 Analýza XML v kontextu komprese XML

Při experimentech s nástrojem XMill a následně i během vývoje vlastního nástroje SXMill jsme došli k závěru, že důležitým bodem komprese XML je analýza vstupních XML dat. Na uvedených příkladech z předchozí kapitoly jsme se snažili ukázat, jak důležité je správně vyhodnotit strukturu dokumentu a identifikovat v ní data. Nesprávné rozlišení těchto dvou elementů může mít nežádoucí vliv na efektivitu komprese, protože bude docházet k nesprávnému odvození sémantických vazeb. Špatná analýza ale může vést až ke ztrátě dat. Vylepšení analyzátoru XML, který by uvedené skutečnosti zohledňoval, je proto tématem dalšího vývoje.

V praktické části této diplomové práce jsme se přesvědčili o tom, že typickým představitelem kategorie XML souborů, u nichž je třeba specifickým způsobem analyzovat strukturu XML, je soubor *wiki.xml*. Podrobnosti jsou k dispozici v kapitole 6, která se věnuje testování.

[6] XMill je specializovaný *XML-aware* kompresor.
[7] Bílá místa.

3 Komprese XML

XML je ze své podstaty velmi výřečným formátem. Veškeré informace jsou v XML uloženy v textové podobě (samotné texty ale i číselné hodnoty, výčtové typy a další specifické informace, které se do textové podoby převádějí serializací dat). Navíc u každé položky se neustále opakuje její sémantická definice, tedy její značka. Ta určuje význam obsahu, který je v ní uzavřen. Například pokud je v XML uloženo několik položek obsahujících informaci o autorech knihy, například pomocí značky <autor></autor>, bude se neustále tato dvojice značek opakovat u každého jednotlivého jména autora. A tím velice rychle roste objem dat v XML souboru.

Zmíněné vlastnosti XML mohou představovat problém při práci s rozsáhlými XML soubory. Jednak je nutné XML data analyzovat, což může představovat zátěž pro výpočetní výkon systému, který s XML pracuje. Problémem může být také velké množství dat, které je nutné archivovat na disku nebo jiném médiu, či přenášet po síti nebo pomalých WAN[8] linkách. A oba zmiňované nedostatky může řešit právě komprese XML.

3.1 Principy komprese

Existuje celá řada algoritmů, které lze použít ke kompresi XML dat. Obecně je možné všechny rozdělit do dvou skupin:

- XML komprese bez podpory dotazování
- XML komprese s podporu dotazování

Algoritmy první kategorie se zaměřují na zmenšení velikosti XML dat s tím, že pokud s komprimovanými daty potřebujeme později pracovat, musíme je nejdříve dekomprimovat jako celek, zpracovat a posléze jako celek znovu komprimovat. V této diplomové práci se zaměřujeme především na tuto kategorii algoritmů. Tyto algoritmy lze dále rozdělit na další dvě podskupiny:

- Komprese XML jako textu (běžná komprese)
- Komprese XML s využitím sémantických informací (XML-aware komprese)

Na kompresi XML jako textu se využívají běžné kompresní nástroje, které v praxi slouží ke kompresi i jiných, nejen textových souborů. Vzhledem k tomu, že XML data jsou v podstatě text, dosahují nejlepších výsledků programy implementující metody specializující se na kompresi textu. Představiteli této kategorie komprese XML jsou například programy *GZip*, *BZip2*[9] či *7-zip*, ale existují i mnohé další. My si dále v následujícím textu popíšeme nejčastější metody komprese, které se napříč různými programy používají.

XML-aware kompresory obecně využívají sémantiku dostupnou v XML datech (pracují se strukturou XML), ale ve svém jádru stále zaměstnávají klasické kompresní algoritmy. Finální komprese tedy probíhá například algoritmem Deflate či BZip2 [14]. Od

[8] Wide-Area-Network, rozsáhlé sítě.

[9] GZip, resp. BZip2 jsou názvy programů, které využívají kompresní algoritmy Deflate, resp. BZip2.

komprese XML jako textu se odlišují především tím, že se snaží specifickým způsobem připravit XML data tak, aby komprese běžnými algoritmy dosáhla lepších výsledků, než jakých se dosahuje při kompresi XML jako textu. Využívá se přitom znalost principů komprese daných kompresních algoritmů.

XML-aware kompresory s podporou dotazování pak zachovávají u komprimovaných dat možnost dotazování. Dotazování mívá různou podporu — liší se rozsah podporovaných dotazů i to, zda je možné data pouze číst nebo i měnit. Tyto algoritmy ve srovnání s doposud popsanými algoritmy dosahují obvykle horších poměrů komprese. Nicméně vedle snížení datové náročnosti mohou odlehčit i výpočetnímu výkonu potřebnému ke zpracování XML dat — vzhledem k nutnosti zpracovat menší množství dat může být zpracování komprimovaných dat paradoxně méně náročné a to i s přihlédnutím na výpočetní výkon, který je vyžadovaný k dekompresi částí XML dokumentu.

3.2 Komprese XML jako textu

Při kompresi XML jako textu se soubor komprimuje jako celek bez ohledu na vnitřní strukturu. Na soubor se pohlíží jako na běžný soubor, nicméně vzhledem k tomu, že se jedná o textový soubor, lze na něj úspěšně aplikovat algoritmy specializující se na kompresi textu. Výhoda toho způsobu komprese je především v jednoduchosti jeho nasazení, protože programů, které se zaměřují na kompresi dat, existuje celá řada. Moderní algoritmy komprese textu jsou navíc velmi efektivní (jak ukazují například výsledky našich experimentů v kapitole 6).

3.2.1 Deflate (gzip)

Deflate je bezeztrátová metoda, která kombinuje kompresi pomocí *LZ77* a *Huffmanovo kódování* [10]. Jedná se o velmi populární metodu komprese dat, což například dokazuje fakt, že její podpora je implementována ve většině moderních vývojových prostředích nebo operačních systémech. Metodu Deflate využívá známý program a formát *ZIP*, to je mimo jiné také obecně zažitý pojem vyjadřující kompresi dat[10] nejen u laické veřejnosti. Deflate je poměrně rychlý algoritmus, což se týká jak komprese, tak i především dekomprese. Jeho nespornou výhodou je i to, že není paměťově příliš náročný.

Proud dat komprimovaný metodou Deflate je tvořen několika bloky, kde každý blok může být uložen jedním z následujících způsobů [10]:

- Blok uložený bez kódování / komprese (hrubá data)
- Blok kódovaný pomocí předem dohodnutého Huffmanova stromu
- Blok kódovaný pomocí Huffmanova stromu, který je součástí bloku

Samotná komprese metodou Deflate probíhá dvoufázově:

1. Pomocí LZ77 jsou odstraněny opakující se řetězce

[10] Slangově se často používá termín „zazipovat soubor“

2. Výstup (1.) je kódován pomocí Huffmanova kódování

Huffmanovo kódování [11] patří do skupiny *statistických kompresních algoritmů*. Statistické metody pracují s četností jednotlivých znaků (nebo jejich skupin) ve vstupním souboru dat. Znakům s vyšší četností jsou přiřazovány kratší kódy (méně bitů, např. nejčastěji se vyskytující znak může být kódován pouze jedním bitem) a znakům s méně častým výskytem jsou přiřazovány kódy delší.

Existují dvě varianty tohoto algoritmu. *Statická varianta* provádí kompresi ve dvou fázích — v první fázi je provedena statistika četnosti výskytu jednotlivých znaků, při které je vytvořen strom kódování, ve druhé fázi dochází k samotnému zakódování vstupních dat s využitím získané statistiky. Výhodou této metody je vytvoření optimální statistiky pro celý vstupní soubor, nevýhodou je pomalé zpracování, protože jsou nutné dva průchody celého vstupního souboru. Další nevýhodou je nutnost uložení binárního stromu spolu s komprimovanými daty. *Dynamická varianta* vytváří statistiku četnosti znaků a samotné kódování během jediného průchodu. To platí jak při kompresi, tak i při dekompresi. Díky tomu není nutné ukládat binární strom spolu s komprimovanými daty. Zároveň je proces komprese rychlejší, protože není nutné vstupní soubor procházet dvakrát, ale na druhou stranu je nutné upravovat strom četností, což samotný proces komprese ve srovnání se statickou variantou zpomaluje. Jelikož má kompresor informace o četnosti znaků pouze u té části souboru dat, kterou doposud prošel, nemusí být přiřazení kódů zcela optimální, čímž může docházet v rámci celého souboru k dosažení horšího výsledku komprese.

LZ77 [23], publikovaný v roce 1977 Abrehamem Lempelem a Jacobem Zivem, je algoritmus patřící do skupiny slovníkových kompresních algoritmů. Algoritmus využívá tzv. posuvné okno — *sliding window* — které obsahuje konec (typicky posledních několik kB) doposud přečtených dat ze zdrojového souboru. Během komprese se algoritmus snaží najít v okně opakující se výskyt části vstupních dat, díky čemuž by bylo možné tento výskyt zakódovat pouze jako *ofset* a *délku* v posuvném okně. Při dekompresi je nutné posuvné okno udržovat stejným způsobem, jak tomu bylo během fáze komprese.

Existují různé varianty algoritmu, lišící se v závislosti na tom, jak kódují výstup. Jako příklad můžeme uvést varianty *LZSS*, *LZH* a *LZB* [2].

3.2.2 BZip2

BZip2 je svobodný, bezeztrátový kompresní algoritmus a také program. Jeho autorem je Julian Seward, který první verzi publikoval v roce 1996. Jedná se opět o poměrně rychlý algoritmus, který ve srovnání s metodou Deflate dosahuje ve většině případů lepších výsledků a jedná se tak celkově o účinnější algoritmus. První verze tohoto algoritmu využívala *aritmetické kódování*, které ale bylo záhy nahrazeno Huffmanovým kódováním. Algoritmus komprimuje bloky dat o velikosti v rozmezí 100 až 900kB (nastavitelné skokově po 100kB). Kombinuje techniky *BWT* (*Burrows-Wheeler Transform*), *MTF* (*Move-To-Front transform*), *Huffmanovo kódování* a *RLE* (*Run-Length Encoding*) [26].

Burrows-Wheeler Transform [16] je transformace známá také pod označením *komprese blokovým tříděním*. Tato transformace ve vstupním souboru nemění hodnotu žádného

symbolu, provádí pouze permutaci jejich pořadí. Pokud vstupní soubor obsahuje opakující se podřetězce, budou po provedení transformace ve výstupu místa, na kterých se budou za sebou nacházet stejné opakující se znaky. A to je předpoklad, díky kterému je možné následně dosáhnout lepšího výsledku komprese. Transformace se provádí setříděním všech rotací textu v tabulce a jako výstup se použije poslední sloupec dané tabulky. Algoritmus 1 znázorňuje pseudokód BWT transformace, inverzní operaci pak zobrazuje algoritmus 2.

Algoritmus 1 Transformace BWT (string s)

```
1: vytvoř tabulku, řádky jsou všechny možné rotace s
2: setřiď řádky abecedně
3: return poslední sloupec tabulky
```

Algoritmus 2 Inverzní BWT (string s)

```
1: vytvoř prázdnou tabulku
2: for i = 1 to délka(s) do
3:     vlož s jako sloupec tabulky před první sloupec tabulky
4:                                   ▷ (prvním vložením se vytvoří první sloupec)
5:     setřiď sloupce tabulky abecedně
6: end for
7: return řádek, u kterého sloupec končí znakem EOF
```

Move-To-Front transform [4], česky *přesuň na začátek*, je metoda, která pracuje na principu nahrazování symbolů vstupní abecedy za jejich indexy do pole symbolů a naopak. Jedná se o reverzibilní transformaci, tzn. že existuje inverzní operace, kterou je možné data vrátit do původní podoby. Proces transformace MTF je následující — každá hodnota vstupu je kódována pomocí indexu, který odkazuje do pole. Toto pole se v průběhu transformace neustále mění. Tedy — v poli je nalezena odpovídající hodnota znaku na vstupu a index této hodnoty je zapsán na výstup. Na začátku je pole uspořádáno podle hodnot (například kódujeme-li jednobajtově, pak 0, 1, ..., 255), první hodnota vstupu je tak vždy zakódována „vlastní" hodnotou. Po zakódování každého znaku je v poli znak přesunut na začátek (odtud název metody).

Reverzní MTF transformace probíhá tak, že se ve výchozím stavu opět začíná s uspořádaným polem (např. hodnoty 0, 1, ..., 255). Dekódování probíhá postupně tak, že zakódovaná hodnota ze vstupu určuje index v poli, kde je uložena hodnota pro výstup. Po dekódování každého znaku dochází k přesunutí tohoto znaku na začátek, stejně jako během procesu kódování.

Pseudokód MTF transformace je zobrazen na výpisu algoritmus 3, pseudokód inverzní operace MTF je pak znázorněn na výpisu algoritmus 4.

RLE (Run-length encoding) představuje jednoduchou formu bezeztrátové komprese. Kóduje vstupní data tak, že opakující se posloupnosti znaků zapisuje jako dvojici

Algoritmus 3 MTF (string s)

```
1: vytvoř pole p (obsahující uspořádané jednobajtové hodnoty (0..255))
2: for all (char z in s) do
3:     v poli p vyhledej index i znaku z
4:     zapiš i na výstup v
5:     v poli p přesuň z na začátek
6: end for
7: return výstup v
```

Algoritmus 4 Invezní MTF (int[] $vstup$)

```
1: vytvoř pole p (obsahující uspořádané jednobajtové hodnoty (0..255))
2: for all (int i in vstup) do
3:     na výstup v zapiš znak z v poli p umístěný na pozici i
4:     v poli p přesuň znak z na začátek
5: end for
6: return výstup v
```

<délka posloupnosti, znak>. Nevýhodou tohoto kódování je to, že výskyt jednoho opakování znaku (jeden bajt) je nutné kódovat pomocí dvojice *<1, znak>* (dva bajty) a tím dochází k nežádoucímu nárůstu dat. Účinnost komprese proto závisí na charakteru vstupních dat.

3.2.3 LZMA

LZMA (Lempel-Ziv-Markov-Chain Algorithm) je vylepšená verze algoritmu Deflate, resp. využívá vylepšenou verzi algoritmu LZ77. LZMA používá stejně jako původní LZ77 *slovník*, ale narozdíl od něj podporuje jeho mnohem větší velikost (až 4GB) a tuto velikost je možné uživatelsky nastavit. LZMA se skládá celkem ze tří součástí — vedle vylepšeného LZ77 pak ještě z kódování *Markov-Chain* a *range kodéru*. Algoritmus dosahuje většinou lepších výsledků než Deflate nebo BZip2, jedná se ale o paměťově náročnější algoritmus, což platí hlavně pro kompresi. Komprese dat je také výrazně pomalejší, nicméně dekomprese je extrémně rychlá a paměťově málo náročná. LZMA je výchozí kompresní metodou formátu *7z* programu *7-zip* [25].

Range kodér, neboli *kódování pomocí intervalu*, využívá k zakódování všech symbolů zprávy pouze jedno číslo — narozdíl například od Huffmanova kódování, které každému symbolu přiřazuje určitou bitovou reprezentaci (nejčastěji se opakujícím znakům co nejméně bitů) a na výstup pak ukládá postupně za sebou odpovídající kódy. Díky této odlišnosti může kódování pomocí intervalu dosáhnout lepších výsledků, než je horní hranice jeden-bit-na-symbol u Huffmanova kódování [15]. Kódování pomocí intervalu je matematicky ekvivalentní k aritmetickému kódování. Podrobnější informace o principu této metody jsou dostupné například v [15].

Markov-chain je matematická metoda pro statistické modelování. Podrobnější informace jsou k dispozici například v [9].

3.2.4 PPM (Prediction by Partial Matching)

PPM [8] je adaptivní, statistická metoda komprese dat, založená na modelech kontextu a předpovídání symbolů. Od svého vzniku, tedy od 90. let minulého století, patří PPM k nejvíce efektivním algoritmům komprese textů v přirozeném jazyce. Jeho historie sahá ještě dále, jeho dřívějšímu rozšíření bránil ale fakt, že je velmi náročný na paměťové prostředky. Jedná se také o časově náročnější metodu a to se týká jak komprese, tak i dekomprese. Existuje několik variant této metody, některé z nich implementují například programy WinRAR nebo 7-zip.

Metoda PPM je založena na modelech [6, 8]. Každý z modelů si udržuje statistiky doposud zhlédnutých symbolů v kontextu předcházejících symbolů. Každý model má určeno, kolik symbolů si bude takto udržovat. Celá metoda PPM pak udržuje několik těchto modelů a každý z nich udržuje různý počet symbolů — od nula symbolů až po maximální počet n, kde hodnota n představuje *stupeň PPM* a značí se typicky *PPM(n)*. Existují také varianty, které nemají pevně stanovený stupeň, nejsou tedy nijak limitovány délkou kontextu, ty se označují *PPM**. Modely slouží k výpočtu předpovědí toho, s jakou pravděpodobností se budou vyskytovat následující symboly. Vypočtená pravděpodobnost se pak používá k zakódování daného symbolu pomocí *aritmetického kódování*. Po zpracování každého symbolu se model upraví tak, aby zachytil právě zpracovaný symbol.

Předpověď pravděpodobnosti pracuje následovně. Pokud je symbol nalezen v nejdelším kontextu, je pravděpodobnost určena jako relativní četnost symbolu v daném kontextu. Pokud není v tomto kontextu symbol nalezen, použije se další nejdelší kontext. Přechod na jiný kontext je indikován zápisem tzv. *escape-znaku* [6]. Tento proces se opakuje do té doby, dokud není nalezena shoda, nebo dokud není k dispozici žádný další kontext. V případě, že již nelze pravděpodobnost určit z žádného modelu, dochází k provedení fixní předpovědi.

Různé varianty PPM se liší v tom, jak řeší problematiku určení pravděpodobnosti doposud neznámých symbolů. Některé varianty těmto symbolům přiřazují konstantně hodnotu 1. Varianta *PPMd*, kterou jsme využili prakticky při implementaci algoritmů komprese XML, navyšuje hodnotu pro každý doposud neshlédnutý symbol o jedna a pravděpodobnost výskytu tohoto symbolu je pak vypočtena jako poměr jedinečných symbolů k celkovému počtu doposud shlédnutých symbolů.

3.3 XML-Aware komprese

Metody komprese, které se přímo zaměřují na XML, využívají sémantické informace uložené v XML datech. Tyto informace jsou v XML přítomny v podobě značek — ty dávají sémantický význam obsahu, který je v nich uzavřen. Základní myšlenka většiny těchto kompresorů je připravit data XML souboru pro běžné kompresní algoritmy tak, aby se využitím vlastností těchto algoritmů dosáhlo efektivnější komprese [1, 6, 14]. Existují

i algoritmy, které se specializují na kompresi struktury XML, přičemž využívají schémat XML (například DTD), nicméně i ty provádí finální kompresi běžnými algoritmy [13].

XML-aware algoritmy lze rozdělit do dvou základních skupin — algoritmy s podporou dotazování a algoritmy bez podpory dotazování.

3.4 Komprese XML s podporou dotazování

Komprese s podporou dotazování komprimuje vstupní XML data a při tom ponechává možnost dále s daty pracovat i v komprimované podobě. Motivace k použití toho principu komprese XML nemusí být čistě z důvodu zmenšení datové náročnosti se zachováním dotazování. Práce s komprimovanými daty a dekomprese pouze vybraných částí XML může být časově méně náročnější, než analýza stejného XML v nekomprimované podobě. Typickými představiteli této kategorie jsou metody *XGrind*, *XPRESS* a *XQzip*.

3.4.1 XGrind

XGrind je dle dostupných materiálů jedním z prvních nástrojů, který se začal zabývat problematikou komprese XML s podporou dotazování. Podrobnější informace jsou k dispozici v [21].

XGrind odděluje strukturu od dat a strukturu kóduje slovníkovým kódováním. U něj kóduje každou značku jako T (tag) a atribut jako A následované jedinečným identifikátorem. Pomocí tohoto identifikátoru se pak odkazuje do slovníku, který obsahuje původní úplný zápis značky nebo názvu atributu. Koncové značky se pak kódují speciálním znakem, XGrind používá konkrétně symbol /. Při dekompresi je koncová značka vždy odvozena z kontextu[11], není tedy nutné pro konkrétní koncové značky vytvářet záznamy ve slovníku.

Kódování struktury je *homomorfní*, to znamená, že komprimovaný soubor je také strukturovaný. Tím pádem je možné prohlížet jej a zpracovávat stejnými nástroji, jaké se používají pro práci s XML v běžné formě [21]. Tento přístup má několik výhod:

- Úpravy dokumentu lze provádět přímo v komprimované verzi.
- Lze využít osvědčené techniky vyvinuté pro práci s XML (analýza či dotazování).
- Komprimovanou verzi XML dat je možné ověřit vůči komprimované verzi schématu XML dokumentu.

XGrind pracuje specificky s výčtovými typy. K jejich identifikaci využívá DTD schéma a kóduje je pomocí $log_2 K$ kódování, kde K je celkový počet hodnot dané domény výčtového typu [21].

Data XGrind komprimuje pomocí bezkontextové komprese[12]. Bezkontextová komprese umožňuje lokalizovat řetězce přímo v komprimovaných datech bez nutnosti je

[11] Vzpomeňme, že well-formed XML dokumenty nedovolují překřížení značek.

[12] Bezkontextová komprese přiřazuje kódy jednotlivým řetězcům tak, že tyto kódy nejsou závislé na aktuální pozici daného řetězce ve zdrojovém souboru dat.

při jejich hledání dekomprimovat. Toho se dosahuje jednoduchým způsobem, protože hledaný řetězec je nejdříve komprimován stejnou metodou, jaká se použila při kompresi vstupního souboru. Takto komprimovaný řetězec se pak přímo hledá v komprimovaných datech.

Proto XGrind využívá konkrétně neadaptivní Huffmanovo kódování. Kontextové[13] algoritmy, jako například LZ77, nejsou pro použití v této situaci vhodné. Pokud by se použila kontextová komprese, došlo by k nárůstu režie nutné k dekompresi každého řetězce před jeho porovnáním s hledanou hodnotou a před provedením samotné dekomprese by musela aplikace vyhodnotit pozici řetězce v souboru a podle toho určit příslušné kódování.

Pro zvýšení efektivity komprese používá XGrind při kódování rozdílné tabulky rozložení četnosti znaků a to zvlášť pro jednotlivé prvky a pro nevýčtové atributy. Tím zohledňuje sémantiku dat, protože jak již bylo několikrát zmíněno, data uložená ve stejné struktuře bývají sémanticky příbuzná.

Architektura XGrind podporuje dotazování v komprimovaných datech v závislosti na typu dotazu. **Dotazy na přesnou shodu**, při kterých se hledá značka nebo atribut přesně se shodující s hledaným výrazem a **dotazy na shodu prefixu**, při kterých se hledá prefix značky odpovídající hledané hodnotě. V obou případech XGrind komprimuje cestu dotazu a predikát dotazu stejnou metodou, jakou použil při kompresi dat. Díky tomu, že se v obou případech využívá bezkontextová komprese, odpovídají kódované hodnoty dotazu přesně hodnotám v komprimovaných datech. XGrind využívá bajtové zarovnání (nikoliv bitové), tzn. porovnávání probíhá vždy na úrovni bajtů, což je mnohem rychlejší než při operacích s jednotlivými bity, nicméně není tak efektivní. Teprve až po nalezení požadovaného prvku dochází k jeho dekompresi. U **dotazů na částečnou shodu** a **rozsah** komprimuje XGrind pouze cestu dotazu. Při postupném průchodu komprimovaných dat pak vyhledává všechny shody hledané cesty a teprve u odpovídajících provádí dekompresi hodnot a vyhodnocení dotazu. Tento typ dotazu je tedy náročnější na vyhodnocení.

Daný způsob dotazování není zcela optimální, jeho nedostatky jsou nastíněny v kapitole 3.4.3.

3.4.2 XPRESS

XPRESS je dalším nástrojem pro kompresi XML z rodiny algoritmů podporujících dotazování. Při jeho návrhu vycházeli autoři především z vlastností nástroje XGrind. XPRESS představil nové, efektivnější metody komprese XML a optimalizoval principy dotazování v komprimovaných datech [18].

XPRESS využívá **automatické odvození datových typů** a provádí jejich efektivní kódování. Autoři se inspirovali XML-aware kompresorem XMill, který také umožňuje efektivně kódovat specifické datové typy (celá čísla atp.), nicméně XPRESS narozdíl od XMill provádí jejich automatickou detekci bez nutnosti uživatelského zásahu. Podporuje celkem šest *sémantických kompresorů*, jejichž přehled zobrazuje tabulka 1. Sémantické kom-

[13] Při kontextové kompresi jsou generované kódy závislé na pozici symbolu ve vstupních datech.

presory *u8*, *u16*, *u32* a *f32* jsou rozdílové kodéry číselných hodnot a kompresory *dict8* a *huff* jsou určené pro kódování textu.

kód	popis kompresoru
u8	kódování celých čísel, kde $max - min < 2^7$
u16	kódování celých čísel, kde $2^7 + 1 < max - min < 2^{15}$
u32	kódování celých čísel, kde $2^{15} + 1 < max - min < 2^{31}$
f32	kódování čísel s desetinnou čárkou
dict8	kódování výčtových dat
huff	Huffmanovo kódování textových dat

Tabulka 1: Přehled sémantických kompresorů nástroje XPRESS

XPRESS komprimuje data bezkontextově — kódování probíhá bez závislosti na jejich pozici v souboru. To opět umožňuje provádět dotazování přímo v komprimovaných datech. Výstup je, stejně jako u XGrind, homomorfní [18].

XPRESS odděluje strukturu od dat. Oproti ostatním algoritmům, popsaných v tomto textu, ale zcela jinak přistupuje k jejímu kódování. XPRESS nevyužívá slovníkové kódování, ale *reverzní aritmetické kódování*. Při něm přiřazuje každé cestě, nebo její podmnožině, jednoznačný interval v rozmezí $<0.0, 1.0)$. Reverzní aritmetické kódování rozděluje celý interval na subintervaly a jednotlivé subintervaly jsou přiřazeny prvkům. Každému subintervalu je přiřazen právě jeden prvek. Velikost každého intervalu je úměrná četnosti prvku (v poměru k celkové četnosti prvků). Podrobnosti postupu výpočtu intervalu jsou k dispozici například v [18].

Vyhodnocení cesty dotazu pak představuje vyhodnocení intervalů. Procesor dotazu převede cestu dotazu (posloupnost prvků od kořene k aktuálnímu prvku) na interval stejným způsobem, jakým to provedl u cest během komprese XML. Poté vyhledá ty prvky, které odpovídají dané cestě podle toho, zda interval cesty dotazu leží v intervalu cesty prvku. Vyhodnocení dotazu pak probíhá pouze u prvků, které odpovídají hledané cestě. Tento postup je tedy efektivnější, než v případě XGrind, který porovnává každou cestu. Nicméně ani tento způsob není zcela optimální, jak je popsáno v následující kapitole 3.4.3.

3.4.3 XQzip

Autoři XQzip si všimli u kompresních algoritmů XGrind a XPRESS několika nedostatků a navrhli další metodu komprese XML s podporou dotazování. XQZip řeší efektivnějším kódováním struktury jak samotnou kompresi, tak i efektivnější dotazování. Pro zvýšení efektivity také pracuje s vyrovnávací pamětí, pomocí které zrychluje provádění opakovaných či podobných dotazů [7].

Nejdříve k problematickým partiím XGrind a XPRESS. XGrind musí při vyhodnocení dotazu procházet celý dokument a u každého zakódovaného prvku či atributu musí porovnávat jeho cestu s cestou dotazu. Pokud se cesty shodují, dotaz se vyhodnotí. Při

provádění dotazu tak musí projít celý soubor a porovnávat jednotlivé cesty. XPRESS řeší tuto problematiku pomocí reverzního aritmetického kódování, které odstraňuje nutnost porovnávat každou cestu, protože se s využitím intervalů vybere pouze odpovídající podmnožina cest. Nicméně i zde je nutné dále vyhodnotit cestu každého prvku této podmnožiny a tato podmnožina může být stále velmi obsáhlá (především u rozsáhlých XML dokumentů, u kterých se často opakuje stejná struktura). XGrind i XPRESS vytváří homomorfní výstup, který je stejně strukturovaný jako vstup, což má přes zmiňované výhody i jednu nevýhodu — pokud se v dokumentu objevuje více dat umístěných ve stejné struktuře, dochází k nárůstu dat samotné struktury, protože se uložení struktury nijak neoptimalizuje [7].

XQzip řeší oba uvedené problémy zavedením struktury *SIT (Structure Index Tree)*, díky které dochází k odstranění duplikovaných struktur. Pomocí hashovacích tabulek pak přiřazuje komprimované bloky dat jednotlivým prvkům této struktury. XQzip dokáže efektivněji vyhodnocovat dotazy, protože nemusí prohledávat celou původní strukturu, ale pouze optimalizovanou v podobě SIT [7]. XQzip podporuje širší škálu XPath dotazů [7], nabízí tak rozsáhlejší možnosti dotazovaní v komprimovaných datech, než jak je tomu u XGrind či XPRESS. Další optimalizační technikou, kterou XQzip využívá, je *buffer-pool*, který v paměti udržuje poslední dekomprimované bloky dat, čímž se částečně odstraňuje režie potřebná pro opakovanou dekompresi dat u podobných či stejných dotazů.

Další podrobnosti o XQzip jsou k dispozici například v [7].

3.5 Komprese XML bez podpory dotazování

Kategorie kompresních algoritmů bez podpory dotazovaní zahrnuje ty algoritmy, které komprimují XML data s využitím sémantických informací (získaných ze struktury XML). Pokud ale chceme s daty pracovat, musíme je nejdříve jako celek dekomprimovat, pak zpracovat a následně opět jako celek komprimovat. Do této kategorie patří například *XMill*, *MHMPPM (XMLPPM)*, *SCMPPM* a *Xml Structure Compression*.

První tři představitelé — *XMill*, *XMLPPM* a *SCMPPM* — pracují na stejném principu. Zpracovávají a připravují XML data tak, aby využili co nejvíce vlastností kompresních algoritmů, které používají k finální kompresi. Mezi ty patří především GZip, BZip2 a PPM.

Metoda *Xml Structure Compression* se nezabývá kompresí samotných dat, ale specializuje se na efektivní kódování struktury. K tomu využívá schémata XML, konkrétně DTD.

3.5.1 XMill

XMill [14] patří mezi nejznámější představitele této kategorie kompresních algoritmů. Jako jeden z prvních použil myšlenku komprimovat XML oddělením struktury od dat a data seskupit podle jejich sémantické příbuznosti. Struktura se kóduje pomocí slovníkového kódování, značky a názvy atributů jsou tedy místo neustálého vypisování nahrazeny odkazy do slovníku. Už jen tento způsob kódování zaručí snížení datové náročnosti kódovaných dat. Data se ukládají odděleně, seskupená podle sémantického významu

(jinak řečeno podle jejich umístění v XML), a pak se komprimují některým z běžných algoritmů. Původní verze XMill podporuje metody Deflate, BZip2 a PPM. Tomuto nástroji se v této diplomové práci věnujeme podrobněji a jeho popisu je věnována speciální kapitola, konkrétně kapitola 4.

3.5.2 MHMPPM (XMLPPM)

MHMPPM (Multiplexed Hierarchical Modeling based on Prediction by Parital Match) [6] představuje další způsob komprese XML založený na obdobném principu, jaký používá XMill. Rozdíl mezi nimi je především ve způsobu kódování struktury a samotných dat.

MHMPPM je založen na použití metody *PPM* [6]. MHMPPM pracuje dvoufázově. V první fázi probíhá kódování vstupního XML metodou *ESAX (Encoded SAX)* a v druhé fázi je výstup ESAX kódován pomocí metody *PPM*.

Během prvních experimentů se autoři zaměřili na odlišný způsob kódování, než jaký používá XMill. Zvolili *ESAX*, který je založen na *SAX modelu*, kdy se jednotlivé události vyvolávané SAX analyzátorem kódují dle následujícího postupu. Jednotlivé události (začátek značky, konec značky, atributy, poznámky, ...) se kódují jako sekvence bajtů. Při jejich tvorbě si kodér a dekodér udržují konzistentní tabulky symbolů. V případě, že kodér narazí na nový, dosud neznámý symbol, přiřadí mu nový kód a ten zapíše do výstupu. Ihned za každý nový kód pak zapíše jeho samotnou hodnotu. Při opakovaném kódování stejného symbolu (řetězce) se pak do výstupu zapisuje pouze daný kód.

Ukázku kódování pomocí *ESAX* si ukážeme na příkladu následující značky [6]:

```
<elt att ="asdf">XYZ</elt>
```

Nyní budeme předpokládat, že kodér již v minulosti narazil na značku <elt> a přiřadil ji kód *10*. Naopak na atribut *att* v tuto chvíli narazí poprvé a přiřadí mu první dostupný volný kód pro atributy, což je *0D*. Výsledek kódování pak bude následující [6]:

```
<elt | att =       | "asdf"     | >  | XYZ          | </elt>
   10| 0D a t t 00 | a s d f 00 | FF | FE X Y Z 00 | FF
```

Takto kódovaný výstup je během druhé fáze komprimován metodou PPM nebo jinou dostupnou metodou komprese dat.

Dalším vylepšením algoritmu bylo použití metody *Multiplexed Hierarchical Modeling*. U původní varianty využíval *ESAX* během kódování pouze jeden model, pomocí kterého postupně zapisoval kódovaná data. Nová metoda používá celkem čtyři modely, mezi kterými přepíná[14] a do kterých zapisuje výstup. Jedná se o modely pro *názvy prvků a atributů*, *strukturu prvků*, *atributy* a *řetězce*. Každý model si pak udržuje svůj vlastní stav, ale všechny čtyři modely sdílejí společný aritmetický kodér [6].

Náš dříve uvedený příklad by se do jednotlivých modelů rozdělil a zakódoval následovně [6]:

[14] Odtud název *multiplexed*.

```
          | <elt | att =            | "asdf"      | >   | XYZ                | </elt>
Prvky:    |10    |                  |             |     | FE                 | FF
Att:      |      | 0D               | a s d f 00  | FF  |                    |
Znaky:    |      |                  |             |     | X Y Z | 00 |       |
Symb:     |      |      a t t | 00  |             |     |                    |
```

Použitím několika od sebe oddělených modelů se dosahuje lepších výsledků predikce pravděpodobnosti symbolů a dosahuje se tak efektivnější komprese. MHMPPM využívá ještě další optimalizační techniky, kdy do jednotlivých modelů vkládá další pomocné informace, aby metoda PPM dosáhla ještě lepších výsledků. Podrobnosti v [6].

3.5.3 SCMPPM

Další metodou komprese XML využívající PPM je *SCMPPM (Structural Contexts Model and Prediction by Partial Matching)*. Kombinuje obecný model pro kompresi strukturovaných dokumentů SCM (Structural Context Model) a kompresní techniku PPM (Prediction by Partial Matching) [1].

SCM je obecný model komprese strukturovaných dokumentů, který vychází z myšlenky, že informace uložené ve stejné struktuře budou mít velmi podobnou slovníkovou distribuci a naopak, že informace uložené v odlišné struktuře budou mít tuto distribuci odlišnou [1]. I zde se tedy předpokládá, že data ve stejné struktuře jsou sémanticky příbuzná a jejich seskupením se bude dosahovat lepších výsledků komprese. Mohou ale existovat i případy, že i data z odlišných struktur jsou sémanticky příbuzná. Představme si informace o knižních záznamech, kde mohou být informace o autorovi knihy a dále informace o lidech, kteří knihy nějakým způsobem revidovali nebo hodnotili — všechny tyto záznamy, byť umístěné v jiné struktuře, obsahují jména lidí. Proto SCM provádí heuristické slučování sémanticky příbuzných skupin do společného kontextu [1].

SCMPPM udržuje PPM model pro každý kontext. Ty se vytváří a udržují pro každou strukturu. Během průchodu skrz XML pak SCMPPM mezi jednotlivými modely přepíná a zapisuje do nich data. Kódování výstupu používá sdílený aritmetický kodéru. SCMPPM obsahuje vždy jeden výchozí model. Pseudokód přepínání modelů je uveden na výpisu algoritmu 5.

SCMPPM pracuje na velmi podobném principu jako XMill. Další podrobnosti o této metodě jsou k dispozici především v [1].

3.5.4 Komprese struktury XML

Doposud popsané metody komprese XML se zaměřují na využití sémantických informací přítomných v XML datech k seskupení sémanticky příbuzných dat a následně jejich kompresi některým z běžných kompresních algoritmů. Strukturu kódují s využitím slovníkového kódování, které ale nijak dále neoptimalizují. *Xml Structure Compression* [13] (komprese struktury Xml) se zabývá především možností efektivněji kódovat strukturu XML a to v případě, že je k dispozici ke konkrétnímu jazyku jeho schéma DTD.

Document Type Definition (DTD) definuje strukturu XML dokumentu tím, že určuje seznam povolených prvků a atributů. DTD lze vložit přímo do XML (inline), nebo jej lze použít pomocí reference na externí soubor [30].

Algoritmus 5 Přepínání modelů MHMPPM

```
1: aktualniModel ← defaultModel
2: while existuje slovo ke zpracování do
3:     slovo ← prectiDalsiSlovo()
4:     for all symbol ze slovo do
5:         kodujDekduj(symbol, aktualniModel)
6:     end for
7:     if slovo je <otevírací značka> then
8:         ulozNaZasobnik(aktualniModel)
9:         aktualniModel ← modelProSlovo
10:    else
11:        if slovo je <koncová značka> then
12:            aktualniModel ← vyzvedniModelZeZasobniku()
13:        end if
14:    end if
15: end while
```

```
<!ELEMENT book (author, title, chapter+) >
<!ELEMENT chapter (title, (paragraph|figure)+) >
<!ELEMENT author (#PCDATA) >
<!ELEMENT title (#PCDATA) >
<!ELEMENT paragraph (#PCDATA) >
<!ELEMENT figure (EMPTY) >
```

Obrázek 3: Příklad DTD [13]

Princip komprese je následující — eliminovat informace definované ve schématu, které se nacházejí redundantně také v samotném XML souboru. Komprese struktury pracuje s DTD schématem, ze kterého jeho analýzou získává charakteristiku XML dokumentů, které jsou vůči němu validní. Tyto informace pak používá k efektivnímu kódování struktury konkrétního XML dokumentu.

Postup komprese si ukážeme na jednoduchém příkladu. Vzorové schéma DTD je znázorněno na obrázku 3, vůči němu validní vzorová XML data jsou pak zobrazena na obrázku 4 (pro správnou kompresi je nutné, aby konkrétní XML data byla vůči DTD validní). Dané DTD schéma popisuje celkem šest prvků. Například u prvku *book* specifikuje, že jeho první podřízený prvek bude prvek *author*, druhý *title* a dále bude následovat jeden nebo více prvků *chapter*. Prvek *chapter* pak jako svůj podřízený prvek musí obsahovat prvek *title*, za kterým musí následovat jeden nebo více prvků *paragraph* nebo *figure*. Podrobné informace o pravidlech zápisu DTD jsou k dispozici například v [30].

Vzhledem k uvedeným pravidlům patrných z DTD, není nutné kódovat některé informace přímo do struktury XML. Například prvek *book* musí obsahovat jako své podřízené prvky *author* a *title* v daném pořadí. Proto je nutné do výstupu zakódovat pouze informaci o počtu následujících prvků *paragraph* nebo *figure*. V našem případě tedy hod-

```
<book>
  <author>Darrell Huff</author>
  < title >How to Lie with Statistics </ title >
  <chapter>
    < title >Introduction</ title >
    <figure ... />
    <paragraph>With prospects of ...</paragraph>
    <paragraph>Then a Sunday newspaper ...</paragraph>
    [ 8 more paragraphs ]
  </chapter>
  <chapter>
    < title >The Sample with the Built−in Bias</title>
    [ 53 paragraphs and 7 figures ]
  </chapter>
</book>
```

Obrázek 4: Příklad XML pro kompresi pomocí DTD [13]

notu 11 (dekadicky), což odpovídá součtu jednoho prvku *figure* a deseti prvků *paragraph*. Analogicky postupujeme pro další prvky. Během fáze dekomprese se k odvození použitých pravidel použije opět dané DTD (které je součástí komprimovaných dat).

Komprimovaný výstup se pak skládá celkem ze tří částí — DTD, kódované struktury a dat. Ke kompresi dat lze použít některou z metod, kterou využívají dříve popsané kompresory XML.

Podrobnosti a příklady komprese struktury jsou dále k dispozici v [13].

4 XMill

XMill [14] je specializovaný nástroj pro kompresi XML dat. Základním principem jeho komprese je zpracování a příprava XML dat tak, aby existující kompresní algoritmy dosáhli lepších výsledků, než jaké dosahují při kompresi XML jako celku. K přípravě dat se využívají sémantické informace přítomné v XML datech — značky, které určují význam obsahu v nich uzavřeném.

XMill je založený na knihovně *zlib* (podporuje tedy kompresi pomocí GZip a BZip2) a využívá několik vlastních *sémantických kompresorů*. Architektura XMill navíc umožňuje rozšíření o další uživatelsky definované kompresory pro komplexní a specifická data různých aplikací [14].

XMill vychází z toho, že slovníkové algoritmy dosahují úspěšné komprese u dat, jejichž slovníkové rozložení je podobné. XMill předpokládá, že data umístěná uvnitř stejné struktury jsou sémanticky příbuzná — například všechna data umístěná ve struktuře <book><author> budou obsahovat jména autorů knihy — a tudíž budou mít podobné slovníkové rozložení [14].

Vedle přípravy dat seskupováním sémanticky příbuzných prvků, které probíhá automaticky na základě struktury XML dokumentu, umožňuje XMill dále ručně doladit parametry komprese uživatelským zásahem dle jeho osobních znalostí komprimovaného XML, především jeho struktury. Jednou z možností je úprava automatického seskupování sémanticky příbuzných dat. Typický je již zmíněný příklad se jmény autorů knižních titulů, ke kterým můžeme přidat další jména lidí, kteří se na přípravě knih nějakým způsobem podíleli. Tyto informace jsou ale většinou k dispozici v jiné struktuře, například v <book><reviewer>, ale protože se stále jedná o jména, bude vhodné je seskupit a komprimovat jako jednu skupinu.

K dosažení efektivnější komprese využívá XMill typově závislé kompresory[15]. Pomocí nich se snaží kódovat specifické informace do efektivnější binární podoby. XMill nenabízí automatickou detekci datových typů[16], ale nechává definici na uživateli. Uživatel tak může určit dvojici *struktura-datový typ* a během komprese je text v dané struktuře analyzován a převeden na odpovídající datový typ v binární podobě. Zcela přirozené je použít tuto myšlenku na číselné hodnoty. XMill nabízí celkem osm vestavěných sémantických kompresorů, jejich seznam je uveden v tabulce 2.

Sémantické kompresory je možné dále kombinovat. K dispozici jsou *sekvenční*, *střídavý* a *opakovaný* kompresor. Sekvenční kompresor lze použít například ke kódování IP adres[17]. Vzhledem k formátu IP adresy (čtyři čísla v rozsahu 0–255, oddělená navzájem tečkou, například 192.168.10.50) je možné zakódovat ji jako čtveřici jednobajtových hodnot. Výčet všech sekvenčních kompresorů je uveden v tabulce 3.

Architektura XMill umožňuje rozšířit nabídku sémantických kompresorů pomocí *uživatelských kompresorů*. Ty se k XMill připojují pomocí specializovaného API[18] rozhraní *SCAPI* (*Semantic Compressor API*). Výhodou použití vlastních sémantických kompresorů

[15] Někdy označované jako *sémantické kompresory*.

[16] Automatickou detekci využívá například nástroj pro kompresi dat s podporou dotazování *XPRESS*

[17] IP adresa identifikuje síťové rozhraní v počítačové síti při použití IP protokolu.

[18] Application Programming Interface, rozhraní pro programování aplikací.

kód	popis kompresoru
t	Výchozí textový kompresor
u	Kompresor pro kladná celá čísla
i	Kompresor pro celá čísla
u8	Kompresor pro čísla menší než 256
di	Kompresor rozdílů celých čísel
rl	Run-length kodér
e	Kodér výčtových typů (enumeration)
...	Kompresor konstant

Tabulka 2: Standardní sémantické kompresory XMill

kód	popis kompresoru
seq	Sekvenční kompresor seq(s1 s2 ...).
alt	Střídavý kompresor or(s1 s2).
rep	Opakovaný kompresor $rep(ds)$, kde d je oddělovač a s sémantický kompresor.

Tabulka 3: Kombinované kompresory XMill

může být přesné cílení komprese na specifickou doménu. Při jejich použití se ale stává komprimovaný soubor závislý na daném kompresoru a není bez něj možné provést dekompresi.

Další podrobnosti o sémantických kompresorech a jejich kombinování jsou k dispozici v [14].

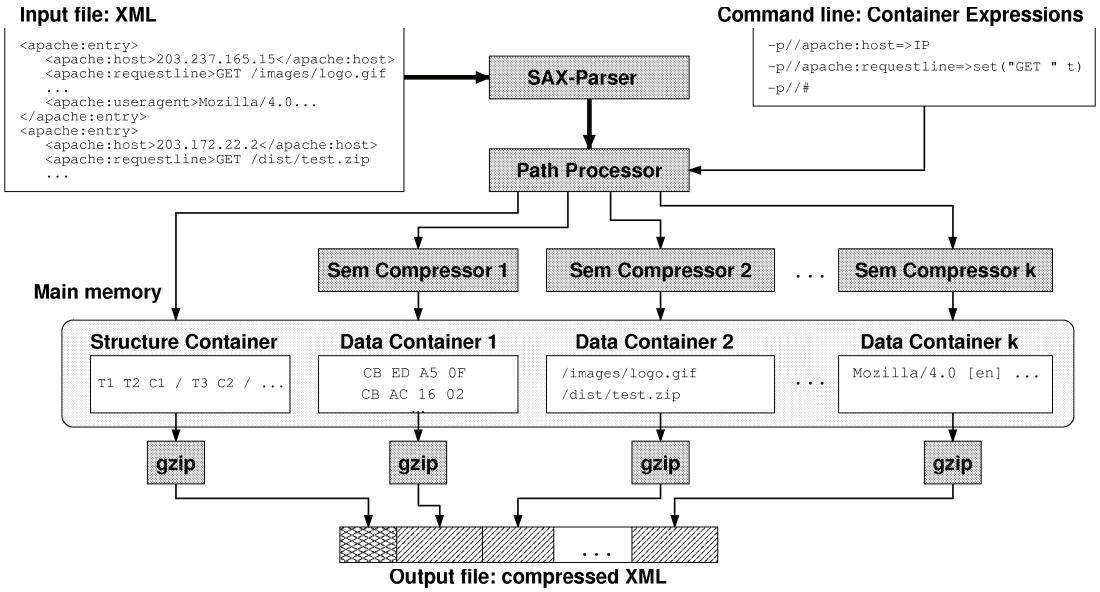
4.1 Architektura XMill

Architektura *XMill* je postavena na třech základech. Prvním je **oddělení struktury od dat**, druhým **seskupení sémanticky příbuzných dat** a třetím **komprese pomocí existujících kompresních algoritmů**. Model architektury je znázorněn na obrázku 5 [14]. Nyní si popíšeme základní prvky architektury.

SAX parser analyzuje vstupní XML data a identifikuje strukturu a obsah XML dat. Analyzované části posílá dále ke zpracování do *Path Procesoru*.

Path procesor je zodpovědný za mapování jednotlivých elementů získaných ze SAX parseru do odpovídajících kontejnerů. Path Processor lze ovlivnit definováním uživatelských výrazů, což je v podstatě vlastní definice sémanticky příbuzných dat.

Kontejnery obsahují data a jsou vytvářeny dle potřeby. Path Processor rozhoduje o tom, zda daný prvek umístí do již existujícího kontejneru, nebo zda vytvoří kontejner nový. Ke každému kontejneru je přiřazen právě jeden sémantický kompresor. Existují také speciální kontejnery, jako například kontejner struktury.



Obrázek 5: Model architektury XMill [14]

Sémantický kompresor je zodpovědný za *analýzu dat* vstupujících do kontejneru a jejich uložení v odpovídajícím datovém formátu[19]. Základním sémantickým kompresorem je *textový kompresor*, který vstupní data vůbec neupravuje a umísťuje je přímo do kontejnerů jako řetězce. Přehled všech sémantických kompresorů ukazuje tabulka 2. Například sémantický kompresor pro celočíselné hodnoty vstupní data analyzuje a převádí na binární hodnotu. V případě, že analýza sémantickým kompresorem selže, předá se hodnota dalšímu, v hierarchii nadřízenému sémantickému kompresoru. Na vrcholu hierarchie je výchozí textový kompresor, tzn. že nejpozději zde se podaří danou hodnotu do některého z kontejnerů umístit.

Komprese dat probíhá nad *paměťovým oknem*[20]. Paměťové okno je kompresorem rezervovaná paměť, do které se postupně zapisují doposud zpracované informace. Velikost paměťového okna je volitelná. Standardní XMill pracuje ve výchozím nastavení s paměťovým oknem o velikosti 8MB, naši implementaci SXMill jsme při experimentech nejčastěji používali s velikostí okna 32MB[21]. Jakmile dojde k zaplnění paměťového okna, přejde se k fázi komprese tohoto okna. Během ní se postupně komprimují jednotlivé části okna — kontejnery. Každý kontejner je komprimován zvlášť do samostatného komprimovaného bloku dat a to pomocí určeného kompresoru (*gzip*, *bzip2*, *ppmdi* atd.). Takto komprimované bloky se postupně za sebou ukládají na výstup. Obecně by komprimova-

[19] Například převod celého čísla z textové podoby do binární.

[20] *Memory window* v terminologii XMill.

[21] Velikost paměťového okna má vliv na účinnost komprese u moderních textových kompresorů, více viz výsledky testování.

Typ	Příklad
(1) Otevírací značky	<items>; <item>; <batters>; ...
(2) Názvy atributů	id; type
(3) Hodnoty (data)	0001; Cake; 1001; Regular; 5001; None, ...
(4) Koncové značky	</batters>; </item>; </items>; ...

Tabulka 4: XMill – přehled částí ukázkového fragmentu XML

ný soubor mohl obsahovat bloky komprimované různými kompresory (za předpokladu, že každý blok je uvozen hlavičkou — což standardně je), nicméně v praxi se využívá v rámci jednoho souboru pouze jedna kompresní metoda.

Paměťové okno je rezervované místo v paměti, do kterého XMill ukládá kódovaná data. Umísťuje se zde hlavička, kontejner struktury, datové a další speciální kontejnery. V okamžiku zaplnění paměťového okna, popř. po dosažení konce vstupního XML souboru, vzniká *běh*, který je ihned komprimován a zapsán na výstup.

Běh[22] je termín, kterým se označují data jednoho paměťového okna. Výstupní komprimovaný soubor se skládá z jednoho nebo více po sobě následujících běhů.

4.2 Příklad kódování ukázkového XML

Při zpracování vstupního souboru provádí XMill několik operací, jejichž výsledkem je výstup série komprimovaných dat. Princip komprese si ukážeme na příkladu XML, který je uveden na obrázku 6. V tabulce 4 je uveden přehled částí vyskytujících se v uvedeném XML. Při zpracování vstupních dat provádí XMill postupně tyto operace:

1. Oddělení struktury od dat.
2. Zpracování dat sémantickými kompresory.
3. Umístění dat do odpovídajících kontejnerů.
4. Komprese jednotlivých kontejnerů (každého zvlášť) zvolenou kompresní metodou.
5. Zápis komprimovaných dat do výstupního souboru.

Body 1 až 3 probíhají během analýzy vstupních dat (odpovědnost mají komponenty *SAX Parser*, *path processor* a jednotlivé *sémantické kompresory*), body 4 a 5 probíhají po naplnění paměťového okna (nebo po dosažení konce vstupních XML dat).

Struktura se kóduje pomocí slovníkového kódování, kde *názvy značek* (1) a *atributů* (2) se označují jako návěští a každé nové, doposud neznámé návěští se přidá na konec slovníku. XMill udržuje speciální *kontejner struktury* (na modelu je vidět zcela vlevo). Do něj se ukládají informace o struktuře XML dokumentu v podobě jednoduchých příkazů.

[22] *run* v terminologii XMill.

```
<items>
  <item id="0001" type="donut">
    <name>Cake</name>
    <ppu />
    <batters>
      <batter id="1001">Regular</batter>
      <batter id="1002">Chocolate</batter>
      <batter id="1003">Blueberry</batter>
    </batters>
    <topping id="5001">None</topping>
    <topping id="5002">Glazed</topping>
    <topping id="5005">Sugar</topping>
    <topping id="5006">Sprinkles</topping>
    <topping id="5003">Chocolate</topping>
    <topping id="5004">Maple</topping>
  </item>
</items>
```

Obrázek 6: XMill – ukázkový fragment XML pro příklad zpracování dat

Tyto příkazy jsou např. *<Vlož značku, id>*, *<Vlož koncovou značku>*, *<Vlož data kontejneru, id>*, kde parametr *id* představuje odkaz na konkrétní návěští, resp. kontejner. U některých příkazů není odkaz vůbec uváděn, protože je patrný z kontextu průběhu komprese či dekomprese (například vložení koncové značky, protože well-formed XML nedovoluje překřížení značek).

XMill využívá sofistikované kódování příkazů. Například pro rozlišení odkazu na návěští od odkazu na kontejner se využívá znaménka u indexu. Zda se má dané návěští zpracovat jako značka nebo atribut, je patrné z průběhu zpracování dat. Veškeré příkazy jsou tak v kontejneru struktury uloženy jako jedna hodnota — číslo. Záporná čísla odkazují na datové kontejnery, kladná čísla pak na značky a atributy. Speciální příkazy (vložení koncové značky apod.) mají přidělenu konstantní hodnotu reprezentující daný příkaz a je jim vyhrazen prostor v rozsahu indexů návěští, kdy čísla do určité hodnoty představují speciální příkazy a nad touto hodnotou se jedná o odkazy do slovníku (absolutní hodnotu indexu získáme po odečtení konstanty počtu definovaných příkazů).

Data (3) se vkládají do samostatných kontejnerů a to v závislosti na jejich umístění ve struktuře XML dokumentu. V kontejneru struktury se po jejich umístění do konkrétního kontejneru vytvoří příkaz odkazující na příslušný kontejner.

Všem *koncovým značkám* (4) je přidělen speciální konstantní kód, který se jako příkaz vkládá do kontejneru struktury. Koncová značka se přidává i na konci každého atributu.

Tabulka 5 ukazuje slovník návěští naplněný tak, jak by vypadal po zpracování ukázkového XML souboru. Rozlišení značky a atributu není nikde uloženo, jak již bylo zmíněno, je patrné z kontextu během zpracování dat. V tabulce je uvedeno pouze pro orientaci.

Během zpracování ukázkových XML dat dojde k vytvoření několika datových kontejnerů. Jejich přehled ukazuje tabulka 6. V tabulce jsou v cestě odlišeny atributy od značek pomocí znaku @. Jak je patrné, pro každou značku nebo atribut dané struktury je vytvořen jeden kontejner, do kterého jsou umísťovány postupně za sebou zpracovaná

Kód	Typ	Návěští
1	Značka	items
2	Značka	item
3	Atribut	id
4	Atribut	type
5	Značka	name
6	Značka	ppu
7	Značka	batters
8	Značka	batter
9	Značka	topping

Tabulka 5: XMill – příklad naplnění slovníku

data. V našem případě neuvažujeme použití žádných speciálních sémantických kompresorů, proto jsou všechna data uložena ve formě textu. Během dekomprese se pak z kontejnerů postupně vyzvedávají jednotlivé prvky a zapisují do výstupního XML souboru. Kontejnery pracují na principu fronty — FIFO.

Tabulka 7 ukazuje obsah kontejneru struktury. Každá hodnota představuje jeden příkaz, který se během dekomprese provede. Kladná čísla odkazují do slovníku návěští (uveden v tabulce 5) a záporná čísla odkazují do datových kontejnerů (uvedeny v tabulce 6). Ve výpisu kontejneru struktury je zástupným symbolem tří teček (...) nahrazeno pětinásobné opakování části `<topping id="XXX">YYY</topping>`. Poznamenejme, že v tomto příkladu jsme nebrali v potaz formátování dokumentu (whitespaces). Pseudokód *K* je příkaz ke vložení koncové značky a pseudokód *PK* je příkaz k vložení prázdné značky (<priklad />). Symbolem # je znázorněn konec kontejneru.

Při pohledu na kontejner struktury je patrné, že jeho data splňují předpoklady pro efektivní kompresi pomocí slovníkových algoritmů (*Deflate*, ...). I přes rozdílná data některých částí vstupního XML je struktura kódována stejnými sekvencemi (zmíněná vynechaná část pětinásobného opakování), protože se zde vyskytují odkazy na stejná místa ve slovníku a odkazy na stejné datové kontejnery. V praxi opravdu komprese kontejneru struktury dosahuje velmi vysoké účinnosti napříč všemi testovanými algoritmy.

4.3 Datový formát XMill

XMill zpracovává vstupní XML data a kóduje je několika různými technikami do výstupního souboru. Základní vlastnosti formátu souboru *XMill v0.8* (v době psaní tohoto textu poslední verze, datovaná březen 2008) si nyní popíšeme.

4.3.1 Uložení čísel, řetězců a seznamů

XMill pracuje pouze s celými čísly (s výjimkou specifických sémantických kompresorů). Formát uložení čísel je podobný, jako se používá u *UTF-8*. Čísla se ukládají pomocí jednoho, dvou nebo čtyřech bajtů. Počet bajtů se určuje podle specifických bitů u prvního

Kód	Cesta	Hodnoty
1	/items/item/@id	0001
2	/items/item/@type	donut
3	/items/item/name	Cake
4	/items/item/batters/batter/@id	1001 1002 1003
5	/items/item/batters/batter	Regular Chocolate Blueberry
6	/items/item/topping/@id	5001 5002 5005 5006 5003 5004
7	/items/item/topping/	None Glazed Sugar Sprinkles Chocolate Maple

Tabulka 6: XMill – příklad datových kontejnerů

1	2	3	-4	K	4	-2	K	5	-3	K	6	PK	7	8	3	-4	K	-5	K
8	3	-4	K	-5	K	8	3	-4	K	-5	K	K	9	3	-6	K	-7	K	...
K	K	K	#																

Tabulka 7: XMill – Příklad obsahu kontejneru struktury

Bity 1.bajtu	Popis
$b7$=0	7 bitové číslo (hodnota $b6$ – $b0$)
$b7$=1 $b6$=0	14 bitové číslo (hodnota $b5$ – $b0$ tohoto bajtu a $b7$ – $b0$ dalšího bajtu)
b7=1 b6=1	30 bitové číslo (hodnota $b5$ – $b0$ tohoto a $b7$ – $b0$ následujících třech bajtů)

Tabulka 8: XMill – uložení čísel bez znaménka (uint32)

Bity 1.bajtu	Popis
$b7$=0	6 bitové číslo (znaménko $b6$, hodnota $b5$ – $b0$)
$b7$=1 $b6$=0	13 bitové číslo (znaménko $b5$, hodnota $b4$ – $b0$ tohoto bajtu a $b7$ – $b0$ dalšího bajtu)
b7=1 b6=1	29 bitové číslo (znaménko$b5$, hodnota $b4$ – $b0$ tohoto a $b7$ – $b0$ následujících třech bajtů)

Tabulka 9: XMill – uložení čísel se znaménkem (sint32)

načteného bajtu (viz tabulky 8 a 9). Data jsou ukládána v ne-intelovském[23] formátu. V tabulkách se odvoláváme na jednotlivé bity v bajtu. Bity číslujeme od 7 (bit s největším významem) do 0 (bit s nejmenším významem). Binární číslo zapsané jako 10000000 (dekadicky 128) má bit $b7$ (bit na pozici 7) nastaven na hodnotu 1, ostatní $b6$ až $b0$ jsou nastaveny na hodnotu 0.

Ukládání čísel bez znaménka, označované jako *uint32*, se řídí logikou uvedenou v tabulce 8. Ukládání čísel se znaménkem, označované jako *sint32*, se pak řídí logikou popsanou v tabulce 9.

Řetězce jsou ukládány s prefixem obsahujícím délku řetězce (uint32), za kterým následují jednotlivé znaky řetězce. Řetězec není ukončen žádným speciálním znakem. Jedná se o datový typ označovaný jako *lstring*.

Seznamy hodnot jsou ukládány ve formátu *prefix + seznam hodnot*. Hodnota prefixu určuje počet elementů v seznamu. Prvky seznamu mohou být čísla (uint32, sint32) nebo řetězce. Seznam řetězců se v XMill vyskytuje poměrně často (například data kontejnerů) a označuje se jako *slist*.

4.3.2 Formát souboru XMI

XMill ukládá komprimovaná data do souboru ve formátu XMI. Ten se skládá ze série komprimovaných bloků a každý blok je komprimovaný některým z podporovaných kompresních algoritmů (gzip, bzip2, ...). Každý komprimovaný blok je uložen včetně hlavičky a signatury kompresoru.

[23] network Byte order

Komprimovaný blok 1	**Komprimovaný blok** 2	**Komprimovaný blok** 3
globální hlavička hlavička běhu 1	datový blok 1 běhu 1 (datový kontejner)	datový blok 2 běhu 1 (datový kontejner)
Komprimovaný blok $i-1$	**Komprimovaný blok** i	**Komprimovaný blok** $i+1$
datový blok j běhu 1 (datový kontejner)	hlavička běhu 2	datový blok 1 běhu 2 (datový kontejner)
Komprimovaný blok $i+2$	**Komprimovaný blok** $i+3$	**Komprimovaný blok** ...
datový blok 2 běhu 2 (datový kontejner)	datový blok 3 běhu 2	...

Tabulka 10: XMill – formát souboru XMI

Id	**Příkaz**
0	Vlož koncovou značku
1	Vlož prázdnou koncovou značku
2	Vlož formátování
3	Vlož formátování atributu
4	Vlož data speciálního kontejneru
5 a více	Vlož odpovídající návěští ($index = id - 5$)
záporné	Vlož data z odpovídajícího kontejneru

Tabulka 11: XMill – příkazy kontejneru struktury

Globální hlavička obsahuje základní informace popisující XMI soubor. Jsou zde uloženy informace o verzi formátu souboru, zda bylo během komprese ignorováno formátování (white-spaces) a informace o *path expressions*.

Hlavička běhu uvádí každý běh. Hlavička mimo popisných informací následujícího běhu obsahuje také data *malých kontejnerů*. Jako malý kontejner se během komprese označí každý kontejner, jehož velikost je menší než stanovená hodnota (výchozí, v aplikaci pevně nastavená, hodnota je 2kB). Uložení malého množství dat přímo do hlavičky vychází faktu, že kompresní algoritmy dosahují horších výsledků komprese nad malým množstvím dat a v některých případech může režie spojená s kompresí objem dat dokonce zvětšit. Hlavička běhu se komprimuje jako jeden proud dat.

Datové bloky obsahují data kontejnerů.

Kontejnery obsahují samotná data. XMill rozlišuje celkem čtyři druhy kontejnerů — kontejner struktury, kontejner speciálních dat (zde se ukládají procesní informace, DTD, CDATA a komentáře), kontejner formátování a datové kontejnery. První tři zmiňované kontejnery se nachází vždy v prvním datovém bloku a tento datový blok existuje v každém běhu.

5 SharpXMill

Pro účely provedení experimentů jsme v prostředí rozhraní .NET Framework, konkrétně v jazyce C#, implementovali vlastní verzi nástroje XMill. Během implementace jsme se soustředili na pokrytí základní funkcionality standardního XMill, proto jsme implementovali podporu pouze textového sémantického kompresoru, což ovšem bylo pro účely našich experimentů zcela dostačující. V první fázi vývoje jsme vytvořili aplikaci kompatibilní s datovým formátem původního nástroje XMill. Dále jsme aplikaci rozšířili o podporu dalších kompresních algoritmů (konkrétně LZMA a PPMdI) a o podporu shlukování kontejnerů.

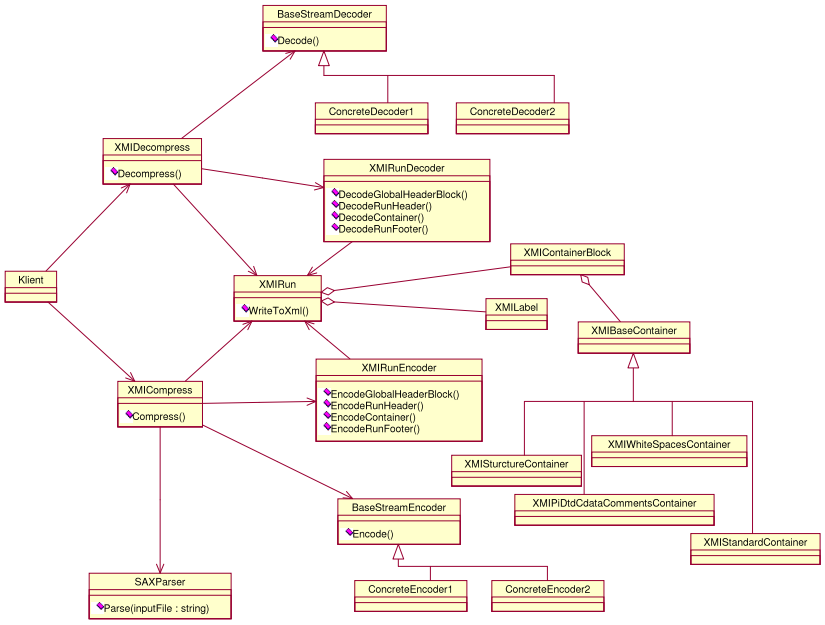
5.1 Návrh architektury SharpXMill

Při návrhu architektury jsme se soustředili na dodržení kompatibility datového formátu a na možnosti rozšíření funkcionality pro účely testování. Plně jsme využili možností objektově orientovaného přístupu návrhu a programování, především dědičnosti a polymorfizmu. Architektura systému je zachycena na obrázku 7. Mezi důležité komponenty architektury v rámci procesu komprese XML patří:

- **SAX Parser** – analyzuje vstupní XML data a jako události reportuje jednotlivé elementy vyšší vrstvě architektury.
- **XMICompress** – komponenta přijímá data od SAX Parseru a je zodpovědná za řízení vytváření aktuálního běhu, kontroly obsazenosti paměťového okna, zakódování běhu do formátu XMill (vytváření jednotlivých, zatím nekomprimovaných, proudů dat) a je zodpovědná za řízení komprese jednotlivých proudů dat zvoleným kompresorem. Je zodpovědná také za reportování různých statistických informací potřebných pro vyhodnocení experimentů.
- **XMIRun** – slouží jako úložiště všech dat aktuálně vytvářeného běhu. Obsahuje návěští, bloky kontejnerů a jednotlivé kontejnery.
- **XMIRunEncoder** – kóduje části XMIRun do jednotlivých bloků (nekomprimovaných) dat. Samotná komprese je spuštěna v okamžiku, jakmile je zaplněno paměťové okno.
- **BaseStreamEncoder** – obecné rozhraní pro kompresi bloku dat. Jednotlivé konkrétní algoritmy (gzip, bzip2, ...) jsou implementovány ve třídách odvozených z této třídy.

Důležité komponenty architektury v rámci procesu dekomprese XML zahrnují:

- **BaseStreamDecoder** – obecné rozhraní pro dekompresi bloku dat. Dekomprimuje data jednoho komprimovaného bloku a předává je dále ke zpracování.



Obrázek 7: Architektura SharpXMill

- **XMIDecompress** – tato komponenta je zodpovědná za řízení průběhu dekomprese dat. Řídí načítání komprimovaných bloků, jejich dekompresi a vytváření objektu XMIRun.
- **XMIRunDecoder** – dekóduje vstupní proud dat formátu XMill a vytváří objekt XMIRun.
- **XMIRun** – stejně jako během fáze komprese, je XMIRun zodpovědný za reprezentaci všech načtených a dekódovaných dat. Dokáže svůj obsah zapsat také do XML souboru.

5.1.1 SAX Parser

Během implementace jsme museli vyřešit problematiku analýzy XML. Zjistili jsme totiž, že XML-aware komprese vyžaduje specifický přístup k analýze XML. Výchozí řešení, které využívalo třídy dostupné přímo v prostředí .NET Framework (konkrétně třídu `XMLTextReader`), jsme museli opustit, protože při práci s XML daty docházelo k nežádoucímu zpracování obsahu. Třída `XMLTextReader` je odvozena od třídy `XmlReader`, tato základní třída slouží v prostředí .NET Framework k analýze a zpracování XML. `XmlTextReader` je velmi komplexní a flexibilní třída, která dokáže mimo základní analýzy XML (rozlišení jednotlivých elementů XML) také provádět nadstandardní XML operace, jako je zpracování odkazů znakových entit, DTD entit apod. Bohužel, pro účely našeho projektu, a obecně pro účely komprese XML, se ukázala tato funkcionalita jako

zcela nežádoucí, protože takto zpracované soubory se neshodovaly s originály, i když docházelo k tomu, že dle specifikace XML obsahovaly stejnou informaci. Některé tyto nadstandardní operace jsme nedokázali deaktivovat, proto jsme museli od použití této třídy upustit. Implementovali jsme proto vlastní SAX parser (`XMLRawReader`), což je třída, která je odvozená od základní abstraktní třídy `XmlReader`.

XMLRawReader během analýzy identifikuje následující základní prvky XML:

- Otevírací značky (např. <barva>).
- Koncové značky (např. </barva>).
- Prázdné značky (např. <barva />).
- Atributy (názvy a jejich hodnoty).
- Formátování (white-spaces).
- Obsah (textové prvky).
- Speciální prvky (komentáře, deklarace XML, CDATA apod.), které vrací jako text.

Veškerý obsah (textové prvky, obsah atributů) předává aplikaci bez jakéhokoliv zpracování nebo změny kódování.

5.2 Podporované kompresní metody

Nedílnou součástí celého systému je podpora běžných kompresních algoritmů. SXMill podporuje kompresi pomocí GZip, BZip2, LZMA a PPMdI. Metody GZip a BZip2 jsou převzaty z knihovny *#ziplib* (*SharpZipLib*) [24]. Podporu komprese LZMA jsme získali díky LZMA-SDK, které je dostupné na webu autora aplikace 7-zip [25]. Podpora PPMdI je zajištěna knihovnou SharpPpmd, která je dispozici na [27].

5.3 SXMill – rozšíření funkcionality XMill

Pro účely testování jsme rozšířili původní XMill o následující funkcionalitu:

- Podpora nových kompresorů LZMA a PPMdI.
- Podpora shlukování kontejnerů s využitím shlukování.

Během testů s kompresí XML jsme experimentovali s využitím metody shlukování dat.

Shluková analýza je proces rozdělení dokumentu se stejnými nebo podobnými vlastnostmi (obsahem) do skupin, které jsou relevantní pro stejné požadavky. Úzce vztažené dokumenty směřuji k tomu, že jsou relevantní vůči týmž požadavkům [17].

Algoritmy shlukování se dělí do několika skupin, podrobné rozdělení shlukovacích algoritmů je například v [22]. My jsme konkrétně využili aglomerativní shlukování. Do

skupiny hierarchických algoritmů pak patří například ještě *divizivní hierarchické shlukování*. Rozdíl mezi nimi je v postupu tvorby shluků.

Aglomerativní shlukování nejdříve vytvoří pro každý objekt jeden shluk. Následně vybere vždy dva nejpodobnější shluky a ty sloučí do jednoho shluku. V posledním kroku jsou pak všechny shluky sjednoceny do jediného shluku. Naopak divizivní hierarchické shlukování vytváří rozklady existujících shluků. Nejdříve je tedy vytvořen jeden shluk, který se postupně dělí tak, až jsou všechny shluky jednoprvkové.

Další podrobnosti o shlukové analýze jsou například v [17, 22].

6 Testování

V této kapitole shrnujeme výsledky námi provedených experimentů. Soustředili jsme se během nich na porovnání účinnosti komprese několika způsobů komprese XML dat. Do testů jsme zahrnuli jak běžně dostupné kompresní programy (RAR [33], 7-zip [25]), tak i XML-aware kompresor XMill [14] v původní a námi modifikované variantě.

6.1 Parametry testování

Jako vstupní data pro experimenty jsme zvolili sadu reálných XML souborů, jejichž seznam uvádí tabulka 12. Testovací soubory jsme před použitím *normalizovali*, abychom dosáhli jednotné formy XML dat — všechny atributy jsou uzavřeny do uvozovek (některé XML soubory používaly apostrofy), došlo k sjednocení formátování (u atributů jedna mezera mezi znakem rovná se (=) zleva i zprava, jednotné odsazení) atp. Účelem normalizace bylo poskytnout vstupní data ve stejném formátu tak, aby výsledky nebyly ovlivněny právě odlišným způsobem zápisu. Výsledkem normalizace je to, že po zpracování normalizovaných souborů našimi nástroji došlo k vytvoření vždy identických souborů (po kompresi a dekompresi, shlukování). Informace o původních a normalizovaných souborech shrnuje tabulka 13[24].

6.1.1 Testovací stroje

Na provedení experimentů jsme použili dva hlavní testovací stroje. Konfigurace stroje A byla následující:

- Intel Core 2 Quad CPU Q6600 2,40 GHz.
- 4 GB operační paměti.
- 32 bitový operační systém Windows Vista Business SP2.

U stroje B se jednalo o virtuální stroj, jehož (virtuální) parametry byly následující:

- Six-Core AMD Opteron 8425 HE 2.08 GHz[25].
- 16 GB operační paměti.
- 64 bitový operační systém Windows Server Enterprise SP2.

Stroj A jsme použili k provedení drtivé většiny testů, stroj B pouze pro některé případy paměťově náročných operací (shlukování a některé testy s kompresí LZMA).

[24] Soubor wiki.xml vznikl spojením několika tisíc jednotlivých XML souborů. Normalizace proběhla automaticky během spojování souborů.

[25] Na virtuálním stroji bylo k dispozici bylo pouze jedno jádro.

Soubor	Velikost	Zdroj
psd_7003.xml	683 MB	XML Data Repository http://www.cs.washington.edu/research/xmldatasets/
dblp.xml	681 MB	XML Data Repository http://www.cs.washington.edu/research/xmldatasets/
wiki.xml	516 MB	Wikipedia XML Corpus http://www-connex.lip6.fr/ denoyer/wikipediaXML/
swissprot.xml	109 MB	XML Data Repository http://www.cs.washington.edu/research/xmldatasets/
nasa.xml	23,8 MB	XML Data Repository http://www.cs.washington.edu/research/xmldatasets/
mondial.xml	1,7 MB	XML Data Repository http://www.cs.washington.edu/research/xmldatasets/
sigmod.xml	0,5 MB	XML Data Repository http://www.cs.washington.edu/research/xmldatasets/
hamlet.xml	0,3 MB	The Plays of Shakespeare in XML http://xml.coverpages.org/bosakShakespeare200.html

Tabulka 12: Sada testovacích XML souborů

Soubor	S_0	S_n
psd_7003.xml	716 853 016 B	716 860 101 B
dblp.xml	714 339 712 B	714 338 879 B
wiki.xml	– B	541 873 094 B
swissprot.xml	114 820 211 B	114 820 211 B
nasa.xml	25 050 288 B	25 054 691 B
mondial.xml	1 784 825 B	1 511 087 B
sigmod.xml	478 416 B	478 416 B
hamlet.xml	288 735 B	288 735 B

Tabulka 13: Testovací soubory XML před a po normalizaci

Metoda	Program	Parametry
GZip	7-zip 4.65	Slovník 32kB, velikost slova 32
BZip2	7-zip 4.65	Slovník 900kB
PPMd (var. PPMdH)	7-zip 4.65	Slovník 256 MB, ppm-order 16, blok dat 4 GB)
LZMA	7-zip 4.65	Slovník 32 MB, velikost slova 64, blok 4 GB
RAR (PPMd var. PPMII)	WinRAR 3.90	Vynucená komprese textu, ppm-order 16, slovník 128 MB (max. dostupná hodnota)

Tabulka 14: Parametry komprese běžnými programy

Metoda	Parametry
xmill/gzip xmill/bzip2 xmill/ppm xmill/lzma	Paměťové okno 32MB, zachování white-spaces

Tabulka 15: Parametry komprese SXMill

6.1.2 Parametry komprese

V první fázi experimentů jsme provedli kompresi XML běžnými nástroji pro kompresi souborů. Parametry jednotlivých testů shrnuje tabulka 14. Druhá část experimentů se soustředila na testy prováděné XML-aware nástrojem SXMill. Parametry těchto testů byly shodné napříč použitými metodami komprese, vše shrnuje tabulka 15.

6.2 Výsledky testování

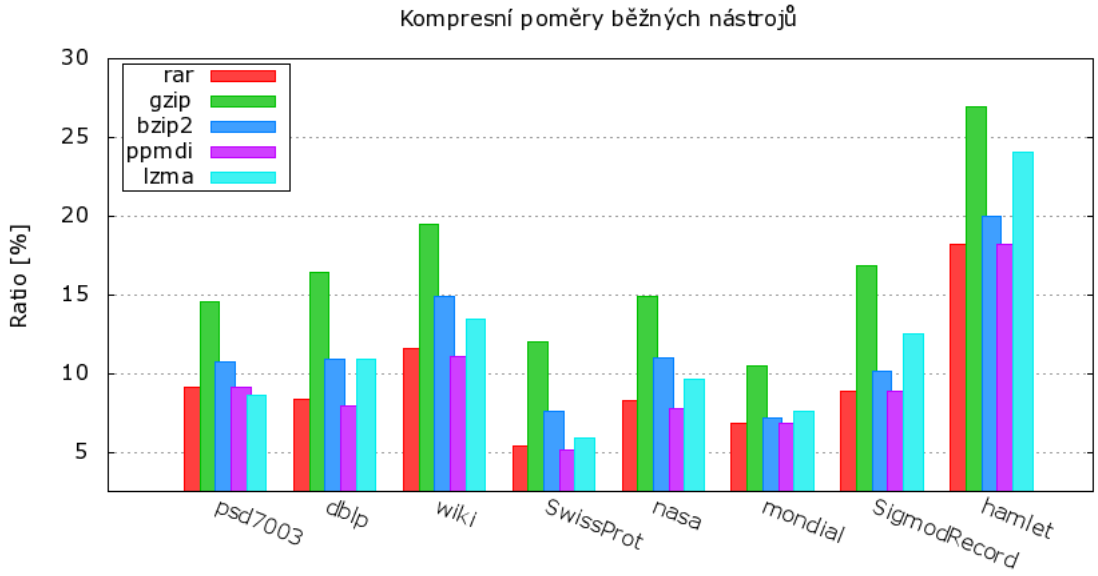
V následujícím textu budeme používat k označení jednotlivých parametrů a výsledků testů notaci, kterou shrnuje tabulka 16. Pokud to má význam, je v přehledu výsledků nejlepší hodnota ze skupiny označena **tučně**. U srovnání poměrů výsledků komprese jsou hodnoty značící zhoršení účinnosti komprese dané metody (hodnoty větší jak 100%) označeny odlišnou barvou.

6.2.1 Komprese XML jako textu

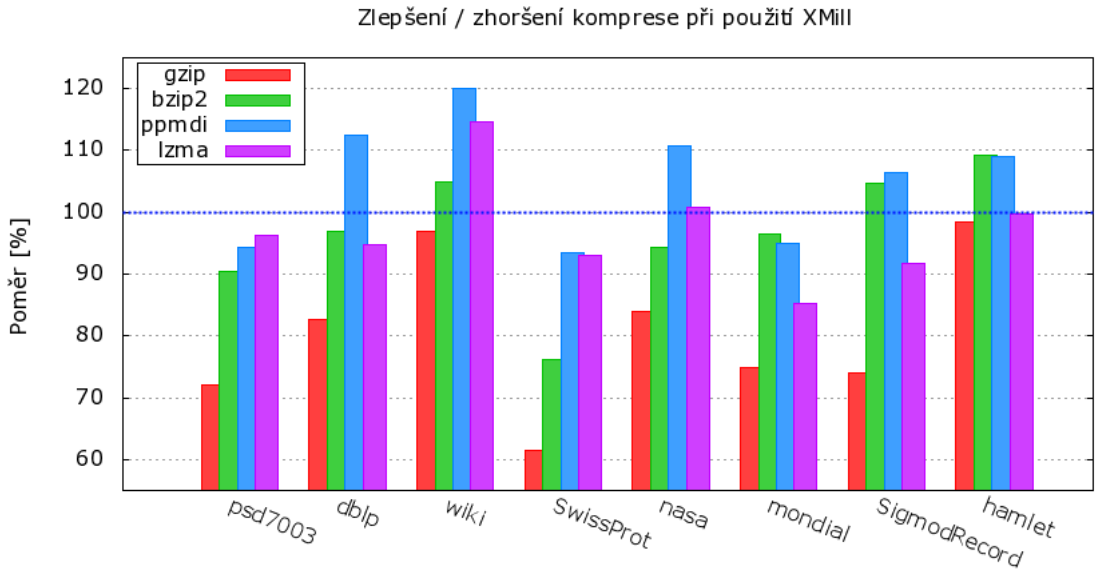
Pro získání referenčních hodnot komprese XML jsme provedli kompresi všech testovaných XML souborů pomocí běžných kompresních nástrojů. Tabulka 17 uvádí absolutní velikosti souborů před a po provedení komprese a tabulka 18 společně s grafem 8 shrnuje kompresní poměry.

Symbol	Význam	Jednotky
S_0	Velikost původního souboru	bajty
S_n	Velikost normalizovaného původního souboru	bajty
CS_{rar}	Velikost souboru komprimovaného metodou RAR	bajty
CS_{gz}	Velikost souboru komprimovaného metodou Deflate (gzip)	bajty
CS_{bz}	Velikost souboru komprimovaného metodou BZip2	bajty
CS_{lz}	Velikost souboru komprimovaného metodou LZMA	bajty
CS_{pp}	Velikost souboru komprimovaného metodou PPMdI	bajty
CR_{rar}	Poměr komprese při použití metody RAR	procenta
CR_{gz}	Poměr komprese při použití metody Deflate (gzip)	procenta
CR_{bz}	Poměr komprese při použití metody BZip2	procenta
CR_{lz}	Poměr komprese při použití metody LZMA	procenta
CR_{pp}	Poměr komprese při použití metody PPMdI	procenta
$CS_{x/gz}$	Velikost souboru komprimovaného metodou XMill/gzip	bajty
$CS_{x/bz}$	Velikost souboru komprimovaného metodou XMill/bzip2	bajty
$CS_{x/lz}$	Velikost souboru komprimovaného metodou XMill/lzma	bajty
$CS_{x/pp}$	Velikost souboru komprimovaného metodou XMill/ppmdi	bajty
$CR_{x/gz}$	Kompresní poměr při použití metody XMill/gzip	procenta
$CR_{x/bz}$	Kompresní poměr při použití metody XMill/bzip2	procenta
$CR_{x/lz}$	Kompresní poměr při použití metody XMill/lzma	procenta
$CR_{x/pp}$	Kompresní poměr při použití metody XMill/ppmdi	procenta
CR_{wavg}	Vážený průměr kompresních poměrů jednotlivých metod	procenta
ΔCR	Poměr poměrů výsledků komprese (vždy srovnání odpovídajících metod komprese) $\Delta CR < 100$ zlepšení, $\Delta CR > 100$ zhoršení, $\Delta CR = 100$ beze změny	procenta
TC_α	Čas komprese $\alpha \in \{rar, gz, bz, lz, pp, x/gz, x/bz, x/lz, x/pp\}$	hh:mm:ss

Tabulka 16: Notace použitá při prezentaci výsledků experimentů



Obrázek 8: Kompresní poměry běžných nástrojů



Obrázek 9: Srovnání výsledků XMill komprese vůči běžné kompresi

Soubor	S_n	CS_{rar}	CS_{gz}	CS_{bz}	CS_{pp}	CS_{lz}
psd7003	716 860 101	65 448 403	104 095 774	76 760 863	65 470 863	**61 473 674**
dblp	714 338 879	59 472 588	117 402 883	77 896 607	**56 460 030**	77 717 175
wiki	541 873 094	62 938 063	105 741 871	80 597 419	**59 888 861**	72 825 200
SwissProt	114 820 211	6 217 395	13 790 955	8 724 363	**5 914 738**	6 775 538
nasa	25 054 691	2 065 876	3 723 344	2 752 252	**1 945 651**	2 415 560
mondial	1 511 087	**103 064**	158 973	108 516	103 241	115 248
Sigmod	478 416	42 513	80 443	48 638	**42 346**	59 945
hamlet	288 735	**52 603**	77 906	57 615	52 586	69 555

Tabulka 17: Absolutní výsledky komprese běžnými nástroji

Soubor	CR_{rar}	CR_{gz}	CR_{bz}	CR_{pp}	CR_{lz}
psd7003	9,12	14,52	10,71	9,13	**8,58**
dblp	8,32	16,44	10,90	**7,90**	10,88
wiki	11,61	19,51	14,87	**11,05**	13,44
SwissProt	5,41	12,01	7,60	**5,15**	5,90
nasa	8,24	14,86	10,98	**7,77**	9,64
mondial	**6,82**	10,52	7,18	6,83	7,63
Sigmod	8,88	16,81	10,17	**8,85**	12,53
hamlet	**18,21**	26,98	19,95	18,21	24,09
CR_{wavg}	8,28	14,78	10,86	**7,84**	9,73

Tabulka 18: Kompresní poměry při použití běžných nástrojů

Z výsledků je patrné, že nejlepších výsledků dosahuje metoda *PPM*, kterou implementuje jak WinRAR, tak i 7-zip. Oba dosahují podobných výsledků, rozdíly plynou z mírně odlišných variant implementovaných algoritmů PPM a ne zcela identických parametrů komprese. Průměrná hodnota kompresního poměru metody PPM byla 7,85%. Nejhorších výsledků dosahoval GZip s průměrnou hodnotou 14,78%. Pokud tyto dvě metody srovnáme, dosahuje PPM téměř o 50% lepších výsledků.

6.2.2 XML-aware komprese XMill

Testovací soubory jsme dále komprimovali pomocí nástroje SXMill. Použité metody komprese jsme volili tak, abychom mohli provést srovnání s běžnými kompresními nástroji (uvedené v předchozí kapitole). V tabulce 19 a v grafu na obrázku 9 jsou uvedeny kompresní poměry jednotlivých metod a poměr vůči výsledku komprese běžných nástrojů. Z výsledků je patrné, že v průměru došlo ke zlepšení u metod GZip ($\sim 19\%$) a BZip2 ($\sim 5\%$), naopak použití XMill spolu s PPM vedlo ke zhoršení ($\sim 7\%$) a při použití LZMA jsme dosáhli srovnatelných výsledků, jako u běžné komprese.

Soubor	$CR_{x/gz}$	ΔCR	$CR_{x/bz}$	ΔCR	$CR_{x/pp}$	ΔCR	$CR_{x/lz}$	ΔCR
psd7003	10,48	72,14	9,67	90,33	8,61	94,22	**8,25**	96,17
dblp	13,59	82,69	10,57	96,96	**8,89**	112,46	10,32	94,82
wiki	18,90	96,87	15,62	104,99	**13,27**	120,09	15,39	114,55
SwissProt	7,40	61,58	5,79	76,25	**4,82**	93,49	5,49	93,07
nasa	12,48	83,97	10,37	94,38	**8,59**	110,66	9,72	100,80
mondial	7,87	74,83	6,93	96,53	**6,49**	94,94	6,50	85,16
Sigmod	12,43	73,92	10,65	104,75	**9,42**	106,44	11,50	91,74
hamlet	26,56	98,45	21,81	109,30	**19,86**	109,02	24,02	99,69
CR_{wavg}	13,54	81,61	11,30	95,62	**9,69**	107,17	10,65	100,30

Tabulka 19: Srovnání XMill a běžné komprese

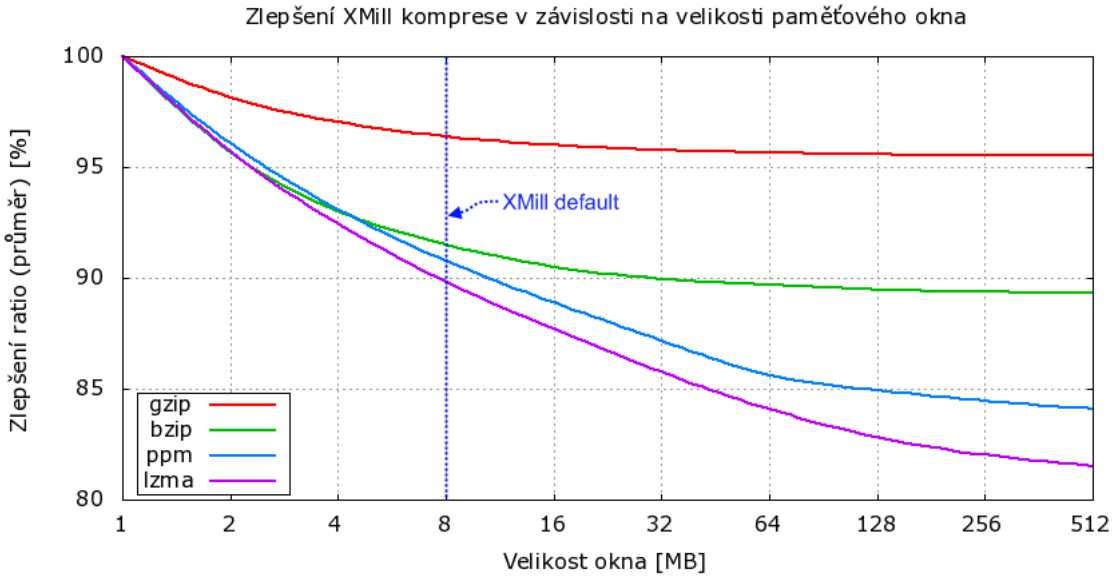
Při podrobnějším prozkoumání výsledků jednotlivých metod komprese je zřejmé, že metoda *PPM* dosahuje velmi dobrých výsledků komprese při kompresi XML jako textu a že použití strukturální komprese (XMill) přináší zlepšení pouze v některých případech a to navíc pouze nepatrné (v řádu jednotek procent). PPM, stejně jako i LZMA, pracují při kompresi s velmi dlouhým kontextem a zvládají při velkých objemech textu generovat efektivní kódování s využitím statistiky pracující s dlouhou historií (jedná se o paměťově náročnější algoritmy). Proto nedochází při použití strukturální komprese, kdy se data tříští do jednotlivých kontejnerů, k výraznému vylepšení komprese.

Navíc po provedení testů bylo ihned zřejmé, že soubor *wiki.xml* je zcela nevhodný pro kompresi nástrojem XMill. Po zjištění výsledků komprese u tohoto souboru jsme proto podrobněji prozkoumali jeho obsah. Zjistili jsme, že se jedná o specifický případ XML souboru. Danou problematiku popisujeme v kapitole 2.5, resp. 2.5.3. V souboru wiki.xml je obsažen souhrn článků, kdy jednotlivé články jsou uzavřeny mezi značkami <article> a </article> (jediná značka na 1. úrovni). Veškerý další obsah představují jednotlivé články. Ostatní značky, které se v nich vyskytují, mají pouze formátovací význam, nikoliv sémantický. Seskupování dat dle těchto informací pak zcela postrádá původní myšlenku a více méně probíhá chaoticky. Mimo to ovlivnil výsledek komprese také fakt, že došlo k vytvoření velmi velkého množství kontejnerů — vzniklo jich více jak 4000.

6.2.3 Paměťové okno XMill

Při kompresi XML pracuje XMill s paměťovým oknem, což je rezervované místo v paměti, do kterého se ukládají doposud zpracovaná data. Po zaplnění tohoto okna jsou data zkomprimována. Původní XMill [14] pracuje s výchozí velikostí tohoto okna 8MB, my jsme při testech pracovali s oknem o velikosti 32MB. Zajímalo nás, jakým způsobem ovlivňuje velikost okna úspěšnost jednotlivých metod komprese. Graf 10 znázorňuje závislost zlepšení komprese na rostoucí velikosti paměťového okna.

Původní XMill byl navržen především pro použití s metodami GZip a BZip2 a jako výchozí hodnotu pro velikost paměťového okna zvolili autoři hodnotu 8MB. Jak je z



Obrázek 10: Zlepšení komprese v závislosti na velikosti paměťového okna

grafu patrné, byla tato hodnota zvolena jako kompromis, při kterém již dalším zvětšováním efektivita komprese neroste natolik, aby se vyplatilo klást větší nároky na paměť. Metody PPM a LZMA ale ukazují, že efektivita jejich komprese roste s větším množstvím současně komprimovaných dat, kdy relativně velký nárůst zlepšení komprese ustává až kolem hranice velikosti okna 128MB, resp. 256MB a dále i za touto hranicí dochází stále k zajímavému zlepšení. Vzhledem k paměťovým nárokům při takto objemných paměťových oknech se ale praktické využití ukazuje jako nereálné a spíše to ukazuje na fakt, že architektura XMill není vhodná pro efektivní využití těchto metod komprese.

6.2.4 Shlukování kontejnerů

Jeden z provedených způsobů optimalizace XML komprese bylo provádění shlukování kontejnerů. Jedná se o metodu, při které před komprimací jednotlivých kontejnerů provádíme shlukovaní dat v nich obsažených. Vzhledem k architektuře XMill je ale nutné po provedení shlukování uložit dodatečné informace o původním rozložení prvků kontejnerů do paty běhu (v komprimované podobě) — tím ale vzniká jistý *overhead*, který negativně ovlivňuje výsledek komprese.

Během provádění praktických testů jsme narazili na dva zásadní problémy:

- Shlukování kontejnerů je v současné implementaci časově velmi náročné.
- Shlukování zlepšuje kompresi kontejnerů, ale overhead celkovou kompresi zhoršuje.

Soubor	**Velikost**	$TC_{x/gz}$	$TC_{x/bz}$	$TC_{x/pp}$	$TC_{x/lz}$
nasa	25 054 691	3:45:26	3:33:57	3:36:22	3:38:05

Tabulka 20: Časová náročnost XMill komprese se shlukováním kontejnerů

α	CS_α	**z toho režie**	ΔCR **bez režie**	ΔCR **s reží**
x/gz	3 477 590	507 785	94,99	111,24
x/bz	2 973 524	441 997	97,45	114,47
x/pp	2 523 919	420 525	97,69	117,22
x/lz	2 645 959	300 488	96,33	108,67

Tabulka 21: XMill komprese se shlukováním kontejnerů

Vzhledem k časové náročnosti testů jsme uskutečnili praktický test pouze na jednom XML souboru (nasa.xml) a u něj jsme provedli kompresi všemi testovanými algoritmy spolu s aktivním shlukováním kontejnerů. Časovou náročnost shrnuje tabulka 20.

Tabulka 21 a graf 11 zobrazují dosažené výsledky. U všech metod komprese došlo díky shlukování kontejnerů ke zlepšení komprese ($\sim$ 3%), ale započtením režie, která musí být uložena u každého běhu, došlo k celkovému zhoršení komprese v rozmezí 8 až 14 %.

Vzhledem k provedení pouze jednoho testu je těžké u této metody učinit závěr. Nicméně je jasné, že bez vylepšení časové náročnosti a navrhnutí efektivnějšího uložení původního uspořádání prvků v kontejnerech, nelze tuto metodu v praxi aplikovat.

6.2.5 Shlukování celých XML souborů

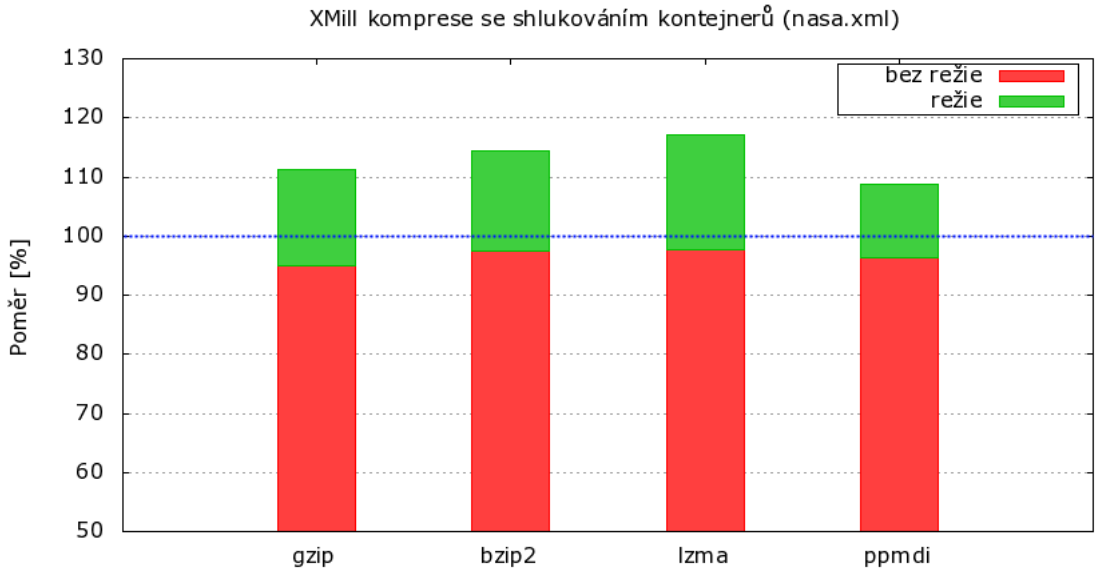
U vybraných XML souborů jsme provedli následující optimalizaci:

1. Rozdělení XML od první úrovně na jednotlivé podsoubory.
2. Provedení shlukování těchto souborů.
3. Složení souborů zpět do jednoho XML dle výsledků shlukování.

Vzhledem k tomu, že jsme prováděli přeházení jednotlivých prvků v XML souboru, není možné v praxi tuto optimalizaci použít obecně na všechny XML soubory. Vhodný XML soubor musí splňovat tato kritéria:

- Přeházení prvků je povoleno (nezakazuje ho například schéma).
- Obsahuje několik prvků na první úrovni, v ideálním případě vhodných pro shlukování (obsáhlejší texty).

My jsme vhodné soubory vybrali ručně, neřešili jsme výběr vhodných souborů algoritmicky. Takto optimalizované XML soubory jsme komprimovali nejdříve běžnými



Obrázek 11: XMill komprese se shlukováním kontejnerů

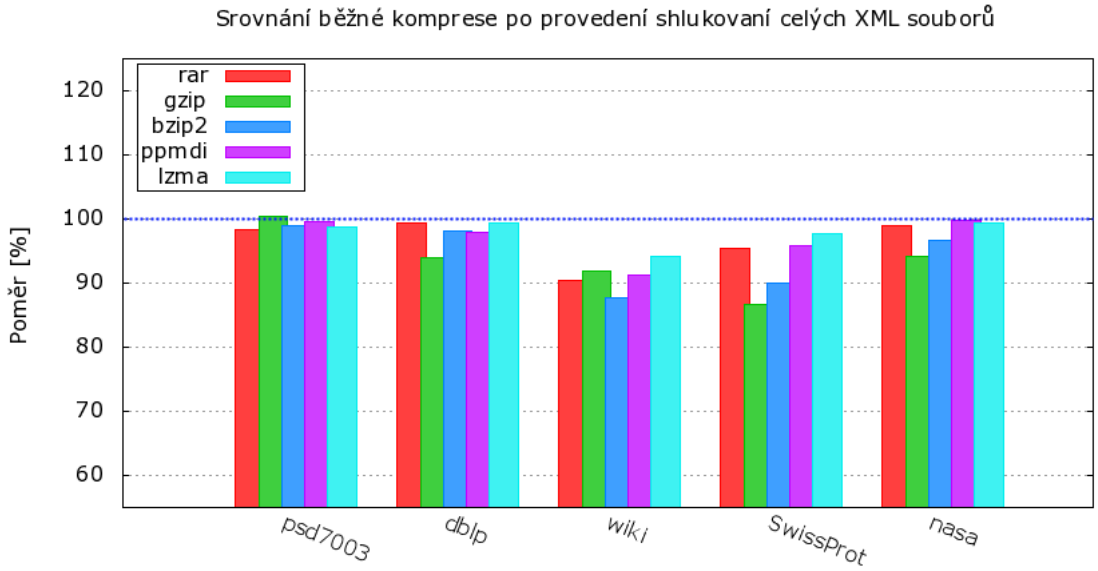
Soubor	Treshold	Prvků/soubor	Souborů	Čas
psd7003	0,05	5	52 506	0:27:08
dblp	0,05	30	71 988	0:52:24
wiki	0,05	1	75 036	0:38:10
SwissProt	0,05	1	50 000	0:12:41
nasa	0,05	1	2 435	0:00:19

Tabulka 22: Parametry shlukování celých XML souborů

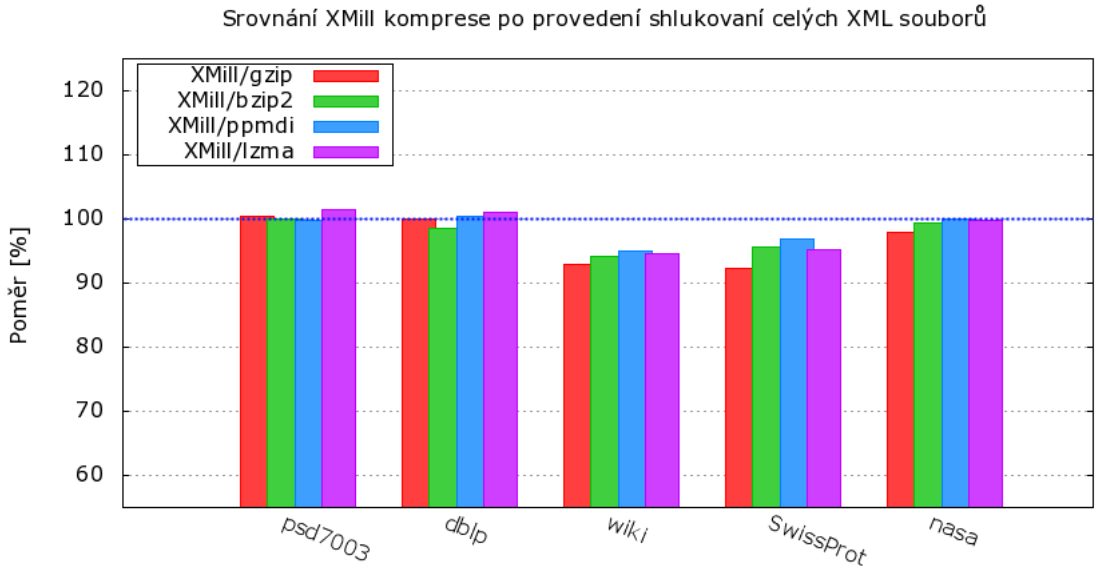
nástroji a pak pomocí XMill ve standardním režimu (bez shlukování kontejnerů). Seznam vybraných souborů a základní parametry shlukování shrnuje tabulka 22.

Výsledky komprese běžnými nástroji celých shlukovaných souborů, včetně srovnání poměru vůči kompresi běžnými nástroji bez provedení shlukování, shrnuje tabulka 23 a graf na obrázku 12. Jak je vidět, až na jeden případ, kdy došlo k nepatrnému zhoršení, mělo shlukování celých XML souborů pozitivní vliv na výsledek komprese. Komprese se v průměru zlepšila (∼3 až 5 %). U souboru *wiki.xml* došlo k téměř desetiprocentnímu zlepšení, což je výsledek daný povahou obsahu tohoto souboru — tento výsledek je v kontrastu s výsledky komprese *wiki.xml* pomocí XMill.

Výsledky komprese pomocí XMill u shlukovaných souborů, včetně srovnání poměrů vůči kompresi XMill u souborů bez provedení shlukování, shrnuje tabulka 24 a graf na obrázku 13. Výsledky ukazují, že u této metody došlo v průměru k mírnému zlepšení výsledků komprese (∼ 1 až 3%). U několika jednotlivých XML souborů ale došlo k mírnému zhoršení.



Obrázek 12: Srovnání běžné komprese po provedení shlukovaní celých XML souborů



Obrázek 13: Srovnání XMill komprese po provedení shlukovaní celých XML souborů

Soubor	CR_{rar}	ΔCR	CR_{gz}	ΔCR	CR_{bz}	ΔCR	CR_{pp}	ΔCR	CR_{lz}	ΔCR
psd7003	9,08	98,34	13,64	100,51	10,51	98,91	8,95	99,52	**8,53**	98,69
dblp	8,19	99,46	16,52	93,92	10,79	98,14	**7,87**	97,99	10,74	99,43
wiki	10,50	90,40	17,94	91,92	13,06	87,79	**10,09**	91,26	12,65	94,14
SwissProt	5,17	95,51	10,40	86,60	6,84	89,98	**4,94**	95,90	5,77	97,71
nasa	8,16	98,93	14,01	94,23	10,61	96,59	**7,75**	99,80	9,58	99,31
CR_{wavg}	8,92	96,53	15,54	**95,24**	11,06	95,28	**8,64**	96,69	10,19	97,73

Tabulka 23: Komprese shlukovaných XML souborů běžnými nástroji

Soubor	$CR_{x/gz}$	ΔCR	$CR_{x/bz}$	ΔCR	$CR_{x/pp}$	ΔCR	$CR_{x/lz}$	ΔCR
psd7003	10,52	100,44	9,68	100,10	8,58	99,69	**8,53**	101,56
dblp	13,58	99,92	10,42	98,55	**8,93**	100,50	10,74	101,05
wiki	17,58	93,01	14,71	94,17	**12,61**	94,98	12,65	94,67
SwissProt	6,83	92,38	5,54	95,60	**4,67**	96,87	5,77	95,29
nasa	12,22	97,93	10,32	99,47	**8,60**	100,01	9,58	99,70
CR_{wavg}	13,19	97,89	11,00	97,80	**9,52**	98,60	10,19	99,26

Tabulka 24: Komprese shlukovaných XML souborů pomocí XMill

7 Závěr

Z provedeného šetření vyplývá, že ke kompresi XML souborů lze využít jak běžné nástroje pro kompresi (archivaci) dat, tak i specializované XML-aware nástroje. Populárním nástrojem v kategorii XML-aware komprese je XMill.

XMill v době svého vzniku dosáhl zajímavého zlepšení u algoritmů GZip a BZip2 tím, že využil strukturální informace přítomné v XML. Seskupením sémanticky příbuzných dat pomáhá slovníkovým algoritmům dosáhnout lepších výsledků komprese. Z námi provedených testů ale vyplývá, že moderní kompresní metody, jako jsou PPM a LZMA, dokáží komprimovat XML soubory jako text v průměru s minimálně srovnatelnou účinností (v implementaci běžných nástrojů), jako uvedené metody GZip a BZip2 při použití strukturální komprese (implementované v XMill). Ukázalo se také, že architektura XMill není příliš vhodná pro nahrazení metod GZip a BZip2 metodami PPM či LZMA, protože nedokáže kvůli tříštění dat do skupin (kontejnerů) naplno využít jejich síly. Využití těchto nástrojů by mělo smysl pouze s velmi obsáhlým paměťovým oknem, což by ale kladlo velké nároky na paměť jak při procesu komprese, tak i dekomprese.

Z provedených testů také vyplynulo, že XML-aware komprese je velice citlivá na analýzu vstupních XML dat. Velice důležitá je správná identifikace a oddělení od sebe struktury od dat. Nesprávné provedení analýzy má vliv na účinnost komprese. Ukázalo se také, že některé XML soubory nejsou vůbec vhodné pro kompresi XML-aware nástrojem XMill a lepších výsledků se dosáhne jejich kompresí jako textu.

Ukázali jsme také, že efektivitu XML komprese lze zvýšit použitím metod shlukování dat. Otestovali jsme shlukování dat v kontejnerech nástroje XMill a také shlukování celých XML souborů a jejich následnou kompresi běžnými kompresory a pomocí XMill. Shlukování kontejnerů mírně jejich kompresi zlepšuje, ale v aktuální implementaci je časově příliš náročné a navíc tak vzniká dodatečná režie, která celkově zabraňuje dosažení zlepšení komprese. Naproti tomu shlukování celých XML souborů u drtivé většiny testovacích souborů vedlo ke zlepšení komprese a to jak při jejich kompresi jako textu, tak i při kompresi pomocí XMill. V průměru jsme dosáhli mírného zlepšení v řádu několika procent, u individuálních souborů bylo zlepšení až o 10 % v závislosti na použité metodě komprese.

8 Reference

[1] J. Adiego, G. Navarro, P. Fuente. Using Structural Contexts to Compress Semistructured Text Collections. *Depto. de Informática, Universidad de Valladolid, Depto. de Ciencias de la Computación, Universidad de Chile, Depto. de Informática, Universidad de Valladolid, 2007.*

[2] T. Bell, D. Kulp. Longest-match String Searching for Ziv–Lempel Compression. *1993.*

[3] M. Brauer, P. Durusaum, G. Edwards, D. Faure, T. Magliery, D. Vogelheim. Open Document Format for Office Applications (OpenDocument) v1.0. *OASIS Standard, 2005.*

[4] M. Burrows and D.J. Wheeler. A Block-sorting Lossless Data Compression Algorithm. *1994.*

[5] Ch. Bussler. B2B Protocol Standards and their Role in Semantic B2B Integration Engines. *Oracle Corporation, 2001.*

[6] J. Cheney. Compressing XML with Multiplexed Hierarchical PPM Models. *Cornell University, Ithaca, 2001.*

[7] J. Cheng, Wilfred Ng. XQzip: Querying Compressed XML Using Structural Indexing. *Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong, 2004.*

[8] J. G. Cleary, I. H. Witten. Data Compression Using Adaptive Coding and Partial String Matching. *1984.*

[9] G. V. Cormack, R. N. S. Horspool. Data Compression Using Dynamic Markov Modelling. *1986.*

[10] P. Deutsch. DEFLATE Compressed Data Format Specification version 1.3. *RFC 1951. 2003.*

[11] D. Huffman. A method for the construction of minimum redundancy codes. *1952.*

[12] J. Kelbel, D. Šilhán. Shluková analýza.

[13] M. Levene, P. Wood. XML Structure Compression. *Birkbeck College, University of London, 2002.*

[14] H. Liefke, D. Suciu. XMill: an Efficient Compressor for XML Data. *Univ. of Pennsylvania, 2000.*

[15] G. N. N. Martin. Range encoding: an algorithm for removing redundancy from a digitised message. *1979.*
`http://www.compressconsult.com/rangecoder/rngcod.pdf.gz`

[16] G. Manzini. The Burrows-Wheeler Transform: Theory and Practice. *1999.*

[17] J. Martinovič. Search in Documents based on Similarity. *Department of Computer Science, Faculty of Electrical Engineering and Computer Science, VŠB – Technical University of Ostrava, 2008.*

[18] J. Min, M. Park, Ch. Chung. XPRESS: A Queriable Compression for XML Data. *Division of Computer Science, Department of Electrical Engineering & Computer Science Korea Advanced Institute of Science and Technology, Taejon, Korea, 2003.*

[19] M. Nicola J. John. XML Parsing: A Threat to Database Performance. 2003.

[20] M. Nottingham, R. Sayre. The Atom Syndication Format. *RFC 4287, 2005*

[21] P. M. Tolani, J.R. Haritsa. XGRIND: A Query-friendly XML Compressor. *Dept. of Comput. Sci. & Autom., Indian Inst. of Sci., Bangalore, 2002.*

[22] M. Vicher. Shlukování pomocí algoritmu COBWEB. *VŠB – Technická univerzita Ostrava, 2010.*

[23] J. Ziv, A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory. 1977.*

[24] .NET Zip Library #ziplib (3.5.2010).
http://www.icsharpcode.net/OpenSource/SharpZipLib/Default.aspx

[25] 7-Zip – domovská stránka (3.5.2010).
http://www.7-zip.org/

[26] BZip2 Manual (3.5.2010).
http://www.bzip.org/1.0.5/bzip2-manual-1.0.5.pdf

[27] Dmitry Shkarin's PPMd Ported To C# by Michael Bone (3.5.2010).
http://users.senet.com.au/ mjbone/Compression.html

[28] Document Object Model (DOM) (3.5.2010).
http://www.w3.org/DOM/

[29] Extensible Markup Language (XML) (3.5.2010).
http://www.w3.org/XML/

[30] Introduction to DTD (3.5.2010).
http://www.w3schools.com/dtd/dtd_intro.asp

[31] Introducing the Office (2007) Open XML File Formats (3.5.2010).
http://msdn.microsoft.com/en-us/library/aa338205.aspx

[32] Overview of SGML Resources (3.5.2010).
http://www.w3.org/MarkUp/SGML/

[33] WinRAR – domovská stránka (v češtině) (3.5.2010).
http://www.rar.cz/

[34] RSS 2.0 Specification (3.5.2010).
http://www.rssboard.org/rss-specification

[35] SAX Project (3.5.2010).
http://www.saxproject.org/

[36] SOAP Specifications (3.5.2010).
http://www.w3.org/TR/soap/

[37] XHTML 1.0: The Extensible HyperText Markup Language (Second Edition) (3.5.2010). http://www.w3.org/TR/xhtml1/

[38] XMill project at sourceforge.net (3.5.2010).
http://sourceforge.net/projects/xmill/

[39] XML Binary Characterization (3.5.2010).
http://www.w3.org/TR/xbc-characterization/

[40] XML-RPC Specification (3.5.2010).
http://www.xmlrpc.com/spec