VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

POROVNÁNÍ VLASTNOSTÍ DISTRIBUOVANÝCH SOUBOROVÝCH SYSTÉMŮ

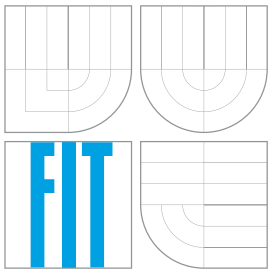
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

VÍTĚZSLAV HUMPA

BRNO 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

POROVNÁNÍ VLASTNOSTÍ DISTRIBUOVANÝCH SOUBOROVÝCH SYSTÉMŮ

COMPARISON OF DISTRIBUTED FILE SYSTEM FEATURES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

VÍTĚZSLAV HUMPA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. IGOR SZÖKE

BRNO 2008

Zadání bakalářské práce/6905/2007/xhumpa00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2007/2008

Zadání bakalářské práce

Řešitel: **Humpa Vítězslav**
Obor: Informační technologie
Téma: **Porovnání vlastností distribuovaných souborových systémů**
Kategorie: Počítačové sítě

Pokyny:

1. Vyberte si několik distribuovaných souborových systémů (DFS) a seznamte se s jejich vlastnostmi a možností nastavení.
2. Vybrané systémy zkompilujte a nainstalujte na testovacím HW.
3. Navrhněte vhodný soubor testovacích úloh.
4. Proveďte sadu testů na všech vybraných DFS. Zaměřte se zejména na rychlost operací čtení/zápis a odolnost proti výpadku.
5. Porovnejte výsledky testů a zhodnoťte vlastnosti zvolených DFS.

Literatura:

- Dle pokynu vedouciho

Při obhajobě semestrální části projektu je požadováno:

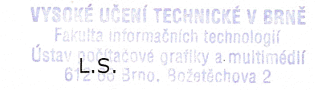
- Bod 1 a alespoň část bodu 2 a 3.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Szöke Igor, Ing.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2007
Datum odevzdání: 14. května 2008



L.S.

doc. Dr. Ing. Pavel Zemčík
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. Pan

Jméno a příjmení: **Vítězslav Humpa**
Id studenta: 78977
Bytem: Dřevařská 870/31, 602 00 Brno
Narozen: 08. 07. 1986, Znojmo
(dále jen "autor")

a

2. Vysoké učení technické v Brně

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
bakalářská práce

Název VŠKP: Porovnání vlastností distribuovaných souborových systémů
Vedoucí/školitel VŠKP: Szöke Igor, Ing.
Ústav: Ústav počítačové grafiky a multimédií
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:
tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☐ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

....

Nabyvatel Autor

Abstrakt

Cílem této práce je porovnání paralelních distribuovaných souborových systémů. Práce je zaměřena na GlusterFS, PVFS 2 a Gfarm FS. Každý systém je popsán po stránce teoretické, včetně možností instalace a konfigurace. Text dále informuje o postupech experimentální práce s těmito systémy a hodnotí výsledky testování, na jejichž základě dále rozebírá vhodnost jejich užití pro praxi na naší fakultě. Všechny testované systémy se ukázaly být dostatečně stabilními pro použití v některé z aplikací na FIT. Jak je v práci rozvedeno, GlusterFS by exceloval v aplikacích současně přistupujících k mnoha velkým souborům (např. systém stahování záznamů přednášek), v přístupu k mnoha malým souborům (např. webserver) by byly všechny systémy vyrovnané a pro práci s menším množstvím větších souborů (např. některé distribuované výpočty) by byly vhodnější PVFS2 a Gfarm FS.

Klíčová slova

paralelní distribuovaný souborový systém, Gfarm Grid File System, GlusterFS, Parallel Virtual File System 2

Abstract

The goal of this thesis is to compare parallel ditributed file systems. Thesis aims at GlusterFS, PVFS 2 and Gfarm FS. There is a description of each filesystem's theory, including general installation and configuration information. The text also informs about experimental work done with each system and evaluates test results which are then used to determine ways of filesystems' possible usage at our faculty. All tested filesystems have shown to be stable enough for use in one of the applications on FIT. As it's described in this thesis, GlusterFS would excel at applications that require simultaneous access to many large files (i.e. recorded lectures download system), all filesystems would give roughly the same performance when accessing many small files (i.e. webserver) and PVFS2 and Gfarm FS would be better suited for work with small number of large files (i.e. some distributed calculations).

Keywords

parallel distributed filesystem, Gfarm Grid File System, GlusterFS, Parallel Virtual File System 2

Citace

Vítězslav Humpa: Porovnání vlastností distribuovaných souborových systémů, bakalářská práce, Brno, FIT VUT v Brně, 2008

Porovnání vlastností distribuovaných souborových systémů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Igora Szökeho

. . . . . .
Vítězslav Humpa
13. května 2008

Poděkování

Děkuji Martinu Tomcovi za spolupráci, Ing. Igoru Szökemu za vedení a Ing. Tomáši Kašpárkovi za odbornou pomoc na této bakalářské práci.

Obsah

Kapitola 1

Úvod

Již od časů prvních počítačů existuje potřeba stálého úložiště dat. V průběhu vývoje moderních počítačů se společně s rostoucím výkonem hardwaru zvyšovala i náročnost softwarových aplikací a s ní potřeba větších, rychlejších a spolehlivějších záznamových zařízení. V současné době jsou možnosti zejména pevných disků již neporovnatelně větší než v minulosti. I přesto existuje množství aplikací potřebujících více prostoru, než je možné uložit na jednom disku.

Je proto vhodné využít současných rychlých síťových technologií a propojením počítačů spojit jejich jednotlivé kapacity a vytvořit větší a rychlejší celek. K tomu je potřeba souborový systém distribuovaný mezi všemi propojenými počítači. Ten zajistí vytvoření jednotného logického jmenného prostoru užívaného klientskými aplikacemi pro práci s takovým uskupením počítačů, které bývá často označováno jako „cluster“.

Standardní distribuované souborové systémy (DFS) jako například NFS nevyužívají více počítačů ke zvýšení své kapacity. Fungují jako samostatné servery a distribuované jsou v tom směru, že se jejich klientská část nachází na jiných počítačích, které k serveru přistupují. Proto existují tzv. paralelní DFS, jejichž serverová část je rozložena na více počítačích.

Společně s kolegou Martinem Tomcem jsme se v rámci našich bakalářských prací[16] zaměřili na srovnání těchto systémů. Ze všech současných paralelních DFS jsme vybrali několik takových, které měly potenciál být využity v rámci potřeb naší školy. Ke zpracování jsme si je rozdělili. Zabýval jsem se především systémy GlusterFS, PVFS 2 a Gfarm Grid FS, zatímco M. Tomec se věnoval systémům Lustre, KosmosFS, Ceph a Starfish.

V úvodu druhé kapitoly je rozvinuta problematika DFS a následovně jsou obecně popsány vybrané souborové systémy. Obsah třetí kapitoly představuje popis experimentální činnosti, kterou jsme s kolegou prováděli s pomocí technických prostředků zapůjčených školou. Poslední kapitolu tvoří přehled některých vlastností vybraných systémů. Poznatky získané během testování byly využity ke srovnání jejich vlastností.

Kapitola 2

Popis distribuovaných FS

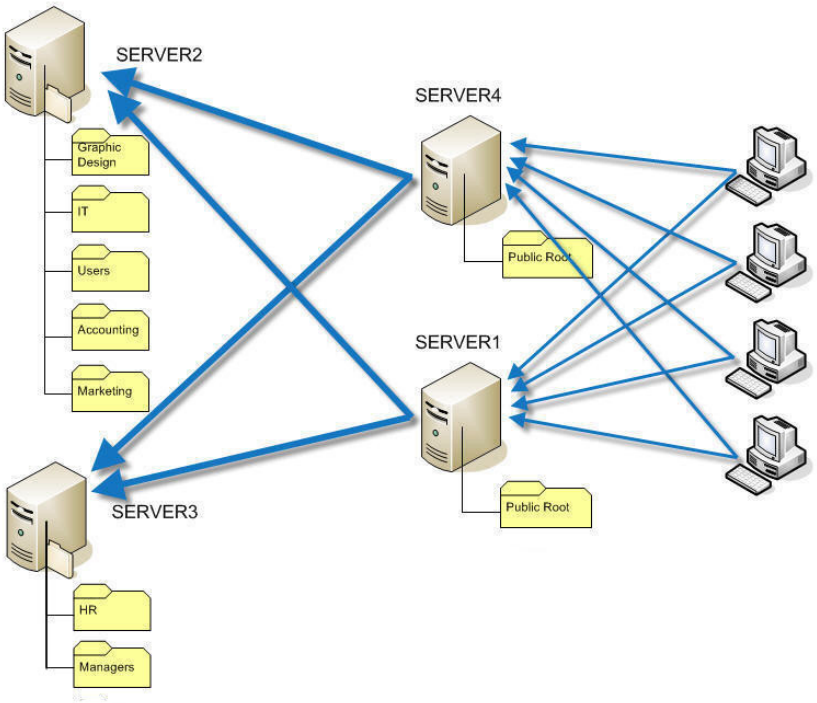
Cílem této kapitoly je seznámení s obecnými charakteristikami distribuovaných FS, kterými jsem se v rámci této práce zabýval. První podkapitola rozvádí pojem paralelního DFS a popisuje jeho typickou stavbu. Nejvíce prostoru je dále věnováno třem DFS, jež byly na základě svých vlastností vybrány pro experimentování na testovacím clusteru. Na začátku každé podkapitoly, patřící některému z těchto DFS, se nachází obecnější úvod seznamující s historií a motivy vzniku jednotlivých DFS. Následuje popis architektury, implementace, způsobu použití apod. Poznatky získané praktickým experimentováním s těmito FS jsou uvedeny v další kapitole.

Poslední podkapitola popisuje další dva paralelní DFS, které však nebyly po konzultaci s vedoucím práce vybrány pro experimentální testování.

2.1 Paralelní DFS obecně

Jak již bylo uvedeno v úvodu, paralelní nebo také clusterovaná DFS, jsou taková, která uložená data tvořící logický jmenný prostor ukládají na více počítačů. Různými způsoby závislými na konkrétní použité implementaci se pak k tomuto clusteru připojují klienti žádající I/O operace. Existují paralelní DFS s nejrůznějšími architekturami, nicméně většina obecně sestává z těchto součástí:

- „*Dataserver*“ – Základní jednotka pro uložení vlastních dat. Instance paralelního DFS může sestávat až ze stovek dataserverů, mezi které jsou rozloženy soubory. Obsah dataserverů dohromady tvoří logický *namespace*.
- „*Metaserver*“ – Server který nakládá s metadaty potřebnými pro nalezení/kompletaci souborů. Metaserver je zpravidla přítomen, nikoli však ve všech paralelních DFS. Některé implementace umožňují provoz vícera spolupracujících metaserverů v jednom clusteru.
- „*Klient*“ – Klientská část zajišťuje připojení paralelního DFS na počítačích, které s ním mají pracovat. DFS je většinou exportovatelný přímo do VFS klientského stroje, někdy je však třeba využít dodávaného API pro použití přímo v potřebných programech.



Obrázek 2.1: Příklad paralelního DFS (Zdroj: [13]); absencí metaserveru také odpovídá architektuře GlusterFS

2.2 GlusterFS

GlusterFS je prvním distribuovaným souborovým systémem z rodiny paralelních DFS, kterým jsem se zabýval. Spolu s prostředím pro spouštění distribuovaných výpočtů GlusterHPC tvoří celek[10] zvaný Gluster (GNU Cluster), což je kompletní distribuce zaměřená na realizaci supervýpočtů pro (nejen) vědecké aplikace. Využívá principu clusterizace, kdy je celkový výkon rozdělen mezi vetší množství standardního hardwaru a nesoustřeďuje se na výkon v rámci jednoho superpočítače. Zatímco GlusterHPC poskytuje rozhraní pro vlastní provedení clusterovaných výpočtů, GlusterFS slouží jako jeho backend k ukládání dat v rámci clusteru.

Prvotní účel GlusterFS je tedy zřejmý, nicméně díky svým vlastnostem je jako samotný DFS vhodný i pro nejrůznější jiná využití. Vhledem k tomu, že je GlusterFS možné nastavit jako systém tolerující chyby hardware, může být využit především tam, kde je potřeba bezpečně uložit značné množství dat. To je nejčastěji použito při realizaci diskového prostoru pro nejrůznější internetové vyhledávače. V případě využití na naší fakultě by tak přicházela v úvahu jednak aplikace původního úmyslu GlusterFS, jednak i jeho použití jako FS pro uložení velkého množství školních multimediálních dat.

2.2.1 Architektura

GlusterFS[1] je snadno a značně škálovatelný. V závislosti na množství počítačů využitých v rámci jedné jeho instance je schopen uložit řádově až petabyty dat - a na druhou stranu může fungovat i jako obyčejný NFS, jako jeden server na jednom počítači, ke kterému se připojují klienti.

[1] Hlavní zdroj teoretických informací o GlusterFS viz [7]

Obecný model

Základem architektury GlusterFS je pojetí klient/server. Vhledem k paralelnosti celého systému může ovšem být v rámci jedné instance GlusterFS serverů spuštěno větší množství. Ke clusteru se pak připojují klienti. Server je implementován jako *glusterfsd*, klienti se k němu posléze připojují pomocí programu *glusterfs* nebo použitím příkazu *mount*. Jako interface zajišťující spojení uvnitř clusteru a mezi clusterem a klienty může být použito rychlého rozhraní *infiniband*, které využívá RDMA[2] přenos. To je užitečné pro tvorbu clusteru, jehož všechny součásti se nacházejí prakticky na jednom místě, výhodou je jednoznačně velká rychlost. Nicméně pro naši potřebu a také s ohledem na hardware testovací sítě využijeme spíše rozhraní *tcp*, které umožňuje spojení součástí clusteru i na větší vzdálenosti – do praxe především pomocí fakultního intranetu.

Za důležitou vlastnost GlusterFS lze považovat tu skutečnost, že jeho součástí není žádný metaserver, který by shromažďoval meta informace o uložených souborech. GlusterFS používá jako svého *storage backendu* standardních FS, jako je nejčastěji *ext3*, ve kterých ukládá celé soubory. Většina ostatních DFS také používá nativní FS, do kterých však ukládá data dávající smysl teprve spolu s jistou formou metadat. Vhledem k této vlastnosti GlusterFS je pak úloha metaserveru zastoupena přímo podnáležícím FS. Výhodou je, že neexistuje kritické místo, při jehož poruše nebo odstavení ze sítě, by došlo ke zneprovoznění celého DFS. Dále to znamená, že GlusterFS neprovádí tzv. *striping*, tedy dělení jednotlivých souborů na části, jež jsou uložitelné na různých dataserverech. To má vliv na možnosti automatické replikace a výkon celého DFS. Tímto aspektem se budu podrobněji zabývat v následující kapitole.

Implementace

Převážná část GlusterFS je implementována pomocí tzv. *translators*. Základem jsou výše zmíněné *glusterfsd* a *glusterfs*, jež však samy implementují jen elementární součásti GlusterFS, především pak společné rozhraní pro použití *translators*. *Translator*, jehož myšlenka byla dle autorů vypůjčena z GNU/Hurd [5]. Jedná se o jistou formu modulu – binárně sdíleného objektu (.so) nahrávaného dynamicky během spouštění *glusterfsd*/*glusterfs*. V praxi je pak většina funkcionality implementována právě formou *translators*, což vytváří jistou modularitu, značně usnadňující rozšiřování možností GlusterFS. Pro představu, např *translator* „cluster/unify“ má na starosti spojení jednotlivých serverů do clusteru a přiřazení plánovače, který se stará o rovnoměrné rozložení zátěže na celý cluster. „Cluster/afr“ zajišťuje automatickou replikaci souborů z jednoho dataserveru na jiný a vytváří tak více kopií souborů pro případ poruchy nebo potřeby výměny některého záznamového zařízení apod. Ne všechny *translators* jsou načteny při spuštění, ale mohou být vybrány a nastaveny v konfiguračním souboru, který popisuji v další kapitole.

2.2.2 Instalace a konfigurace

Glusterfsd server může být přeložen a spuštěn na jakémkoli operačním systému, který podporuje standard POSIX. Klient glusterfs na druhou stranu vyžaduje FUSE pro export instance GlusterFS do jmenného prostoru daného počítače. Dle vývojářů jsou tedy podporována a otestována jádra systémů Linux, FreeBSD, OpenSolaris a MacOS X.

[2] Remote Direct Memory Access[9] – přímý přístup do paměti jiného počítače

Protože se jedná o program pod licencí GPL, je možné stáhnout si zdrojový tarball. Server *glusterfsd* nemá v poslední verzi žádné závislosti [3], klient oproti tomu potřebuje buď *devel* balíček, nebo přeložené a nainstalované *fuse*.

Překlad a instalaci GlusterFS provedeme posloupností příkazů:

```
$ ./configure
$ make
$ make install
```

Výše uvedené přeloží a nainstaluje do systému server i klienta. Volba `--disable-fuse-client` u `./configure` omezí instalaci jen na server a `--disable-server` jen na klienta.

Instalace tímto způsobem je nicméně zdlouhavá, zejména když má budoucí cluster sestávat z většího množství počítačů. GlusterFS je proto k dispozici i předkompilovaný ve formě RPM[4] a DEB balíčků, čímž je usnadněna instalace na Linuxových distribucích založených na Red Hat a Debian. Pokud dodávané balíčky nebudou přímo aplikovatelné pro zvolený systém, je od verze 1.3.pre6 možné u Linuxových distribucí se správou balíčků založenou na RPM si takový balíček vyrobit „na míru“:

```
rpmbuild -tb glusterfs-1.3.pre6.tar.gz
```

Tento příkaz přeloží GlusterFS a vytvoří balíčky pro server i klienta, které je pak možno již přímo použít na ostatních počítačích clusteru, zejména pokud je prostředí na celém clusteru uniformní, což je zajisté častý případ.

Před spuštěním *glusterfsd* i *glusterfs* je nejprve nutné editovat konfigurační soubor zvaný *Volume Specification*, ve kterém je třeba upřesnit informace o konkrétní instanci GlusterFS. Po instalaci se tento soubor nachází obvykle ve `/etc/glusterfs/glusterfs-server.vol`, resp `glusterfs-client.vol`. V případě, že budeme spouštět server, je třeba uvést jakou instanci bude obsluhovat, kam má ukládat soubory, druh spojení, naslouchací port a podobně. Příklad konfiguračního souboru použitého na našem testovacím clusteru uvedu v příští kapitole. Formát konfiguračních souborů je přesně specifikován a jeho popis se nachází na wiki GlusterFS [7].

2.3 Parallel Virtual File System 2

PVFS je ze zmíněných paralelních DFS systémem se zatím nejdelší historií. Ke konci devadesátých let byl prvním skutečně použitelným clusterovým DFS využitelným na Linuxových počítačích. Jako první implementoval pokročilé vlastnosti, které jsou dnes již běžné mezi moderními DFS, především těmi, kterými se zabývá tato bakalářská práce. Do poloviny prvých let tohoto století se ovšem dočkal kompletního přepisu[14] a v roce 2004 byla vydána verze 1.0 jeho následovníka PVFS.

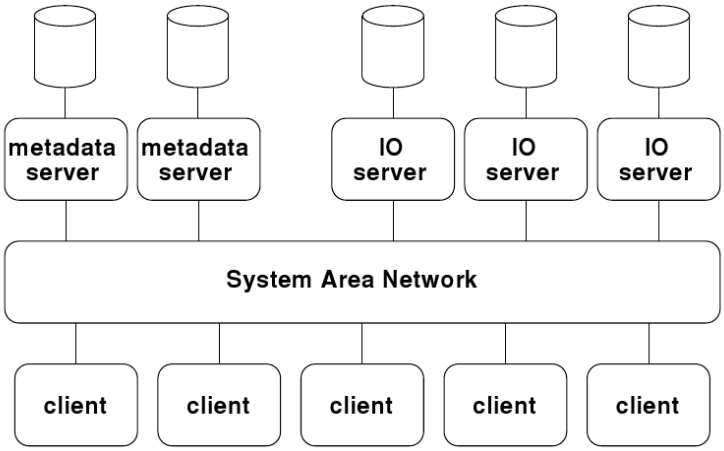
Důvodů ke kompletnímu přepisu bylo dle autorů mnoho. Zejména pak zdůrazňují, že kód byl "a mess", a příliš spoléhal na některé technologie jako *BSD sockets*, se kterými přímo pracoval, čímž byla omezena použitelnost. Snahou PVFS je být v těchto ohledech více abstraktním, nepracovat přímo se síťovými technologiemi poskytovanými OS, ale činit tak přes univerzální mezivrstvu. Vylepšení je mnohem více, obecně vytváří DFS, který ve své druhé verzi lépe využívá možností současných Unixových OS a který by měl být použitelný i mnoho let do budoucna.

[3] Samozřejmě kromě základních prostředků pro překlad jako gcc a gnumake, které jsou v drtivé většině případů na unixových systémech standardně k dispozici

2.3.1 Architektura

Architektonicky[1] se PVFS v mnoha ohledech podobá GlusterFS, samozřejmě s několika podstatnými rozdíly. Především jeho součástí je metaserver. PVFS využívá nativních FS podobně jako GlusterFS. Na druhou stranu neukládá soubory vcelku, ale provádí jejich rozdělení na menší části, známé často jako *chunks* nebo *stripes*. Je třeba, aby jistá komponenta vedla informace o metadatech, s jejichž pomocí je pak možné nalézt celý soubor. K tomu slouží metaserver.

Dále PVFS zcela záměrně deklaruje[14], že není striktně kompatibilní s normou POSIX. Teoreticky jen málokterý FS nebo DFS se této normy stoprocentně drží, nicméně vývojáři se v některých bodech od POSIXu vzdalují více. Podle autorů je POSIX příliš omezující a rozhodnutím se ho přesně nedržet bylo možno přistoupit k některým způsobům implementace, jež přinášejí velké výhody v oblasti výkonové a zejména v otázkách spolehlivosti a konzistentnosti DFS. Bývá především zdůrazňováno, že díky tomu může PVFS být zcela bezstavové, což je faktor, který velice pozitivně ovlivňuje možnosti obnovení systému po výpadku.



Obrázek 2.2: Architekura PVFS 2 [1]

Jako příklad nestavovosti je možné si představit situaci, ve které potřebuje uživatelská aplikace otevřít soubor. Jakmile DFS požadovaný soubor nalezne, tak patřičný server stavového DFS kromě potvrzení klientovi si ve své paměti zaznamená, že dotyčný soubor byl otevřen. Tím vzniká stav a problémy spojené s jeho obnovením, pokud dojde k výpadku. Z hlediska PVFS pak v podstatě stavy jako otevřený soubor apod. neexistují, vytváří je čistě abstraktně klientská část PVFS. V případě výpadku serveru není poté nutné žádnou tabulku stavových informací znovu sestavovat, což obnovení usnadňuje.

PVFS není chybově tolerantní - tedy nevytváří zrcadlové kopie souborů jako předchozí DFS v případě poruch hardwaru. Jednoduchost obnovení po výpadku, umožněná bezstavovostí, toto kompenzuje alespoň v případech, kdy nedojde ke ztrátě dat, ale pouze k dočasnému odstavení pevného disku a podobně. Jsou-li data ukládaná v PVFS důležitá, je možné skutečnou chybovou toleranci nahradit alternativními metodami, jako využitím RAID polí na počítačích v clusteru apod.

Implementace

Narozdíl od GlusterFS využívá PVFS ke správě souborů metadata ukládaná v DB na počítači, na kterém běží metaserver. Nicméně implementačně je vše vyřešeno na první pohled poněkud jednodušeji. Programově PVFS sestává pouze ze dvou částí:

- *pvfs2-server* – hlavní serverový proces PVFS. Tento program/proces je server, který je schopen fungovat jako filesystem server obsluhující počítač aktivně přispívající do clusteru - a také i jako metaserver. V PVFS1 existoval pro metaserver zvláštní proces *mgrs*, v PVFS však byly tyto dva sloučeny. *Pvfs2-server*, může fungovat pouze jako file server, či může fungovat pouze jako metaserver a především může v jednu chvíli zastávat obě role. V PVFS1 býval v clusteru přítomný vždy jen jeden metaserver *mgrs*, nyní však může být součástí clusteru více *pvfs2-server* procesů, které zastávají (i) roli metaserveru a vzájemně spolupracují. Zátěž je tedy rozmělněna a metaserver je eliminován jakožto bod, při jehož výpadku je odstaven i celý DFS. Je-li *pvfs2-server* konfigurován jako metaserver, je potřeba, aby na stejném počítači běžela i databáze pro uložení metadat, kterou je *Berkeley DB*.

- *pvfs2-client* – klientská část PVFS, která musí běžet na těch počítačích, ze kterých chceme k PVFS přistupovat. Na linuxových OS je vlastní dostupnost PVFS zajištěna pomocí specializovaného modulu jádra, a tak je možné jej připojit do systému souborů počítače. Modul komunikuje s *pvfs2-client*, který zajišťuje vlastní síťovou interakci s PVFS clusterem. Na jiných unixových systémech, kde není k dispozici modul jádra, je možné z uživatelských aplikací využít API rozhraní MPI-IO [4] , vytvořené navrchu API knihovny *libpvfs2*.

Výše jsem se zmínil o tom, že PVFS používá abstraktní rozhraní místo přímé práce s *UNIX sockets*. Konkrétně využívá rozhraní BMI (Buffered Messaging Interface), které je schopné zastřešit více možných komunikačních rozhraní. V současné době existují implementace BMI pro TCP/IP, *Myricom GM*[5] a *infiniband* - PVFS tedy podporuje spojení svých součástí s kterýmkoli z těchto rozhraní.

2.3.2 Instalace a konfigurace

PVFS není vydáváno ve formě předkompilovaných balíčků, je třeba si ho stáhnout jako archív se zdrojovými kódy. Následovné přeložení/instalace je pak pouze záležitostí spuštění `./configure`, `make` a `make install`. Pokud chceme vytvořit modul jádra v systémech Linux, je třeba ke `./configure` připojit parametr *-with-kernel=* následovaný cestou ke zdrojovým souborům jádra.

Pro konfiguraci PVFS je připravena utilita *pvfs2-genconfig*, což je interaktivní skript, který uživatele provede konfigurací všech proměnných. Skript je možné spustit na libovolném počítači, na kterém byl PVFS nainstalován - vznikne soubor `/etc/pvfs2-fs.conf`, který je posléze třeba rozkopírovat na všechny počítače, jež budou součástí clusteru, tedy kde poběží *pvfs2-server*. Jakmile je toto dokončeno, rozběhnutí PVFS clusteru se již pouze záležitostí spuštění programu pvfs2-server.

[4] MPI znamená *Message Passing Interface*, více informací viz [6]
[5] GM Message Passing System[2]

Před spuštěním na klientských počítačích je potřeba vytvořit soubor `/etc/pvfs2tab`, který je formátem obdobou standardního `/etc/fstab`. Do něj[6] je pak nutné vložit řádku pro každou instanci PVFS, ke které se budeme chtít připojit, podobnou:

```
tcp://testhost:3334/pvfs2-fs /mnt/pvfs2 pvfs2 defaults,noauto 0 0
```

Je-li klientským počítačem Linux s nahraným modulem jádra, je možné přímo využít příkazu `mount /mountpoint`. PVFS pak bude vyexportován do daného `/mountpoint`. V případě, že OS není Linux, nebo pokud nevyužíváme modulu jádra, ale rozhraní MPI-IO, `/etc/pvfs2tab` vytvořit musíme a v místě `/mountpoint` nebude PVFS vyexportován, nicméně API použité v aplikacích se bude chovat tak, jako by tomu tak bylo.

2.4 Gfarm Grid File System

Dalším z rodiny paralelních DFS s chybovou tolerancí je Gfarm Grid File System, zkráceně Gfarm. Byl vyvinut v Japonsku jako ukázka „Grid Datafarm“ architektury, která, dle přání autorů[11], má být alternativou NFS v situacích, kdy je potřeba mnohem větší, spolehlivější a rychlejší souborový systém.

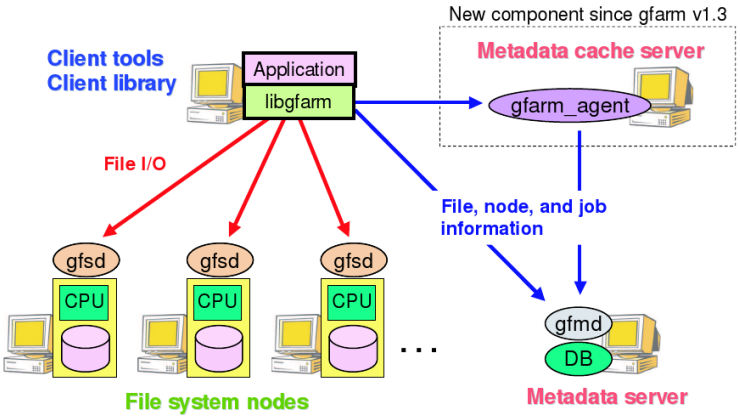
2.4.1 Architektura

Svou hlavní myšlenkou a některými vlastnostmi se Gfarm podobá spíše PVFS, automatickou replikací, kterou PVFS nemá, pak částečně i GlusterFS. Architektonicky se však jedná o systém v mnohém odlišný od obou. Opět se nejedná čistě o systém typu klient/server, ale potřebné role jsou rozděleny mezi více prvků. Pokud považujeme klienta za součást FS, je pojetí Gfarm čtyřstranné, jinak trojstranné. Každá součást Gfarm („node“) bývá standardně spouštěna na zvláštním počítači, nicméně nic nebrání tomu, aby několik - nebo třeba i všechny typy node byly spuštěny na jednom stroji. Gfarm sestává z [17]:

- *„Filesystem node“* – souborový server. Jedná se o základní souborovou jednotku, která je spuštěna na každém počítači clusteru, který přispívá diskovým prostorem do konkrétní instance Gfarm. Na filesystem node běží program (démon) *gfsd*, který má na starosti zpracování operací se soubory nebo jejich částmi, které jsou uloženy na patřičném počítači. Dále zajišťuje autentizaci uživatelů k němu připojených, vytváření replik souborů a koordinuje svou činnost s informacemi, které mu poskytuje metaserver.

- *„Metadata server node“* – server obstarávající metadata. Gfarm ve *filesystem node* využívá nativních FS podobně jako PVFS. Soubory tedy neukládá vcelku, ale provádí jejich rozdělení na menší části. Tyto části jsou poté dále replikovány na další nody, čímž je docílena jednak jistá úroveň ochrany před výpadky hardwaru, dále i optimalizace zátěže Gfarm vytvářením více kopií u souborů často používaných, než u těch, které jsou méně potřebné. K tomu všemu je nezbytný tento server, který uchovává metadata – informace o místech uložení jednotlivých *stripes* – a řídí paralelní procesy související s rozložením zátěže. Metaserver node Gfarmu, *gfmd*, je spuštěn vždy na jednom počítači a obsluhuje celou instanci Gfarmu. Jako databázový backend pro uložení vlastních dat využívá volitelně buď LDAP nebo PostgreSQL server běžící na stejném počítači jako metaserver.

[6] Je možné vložit tuto řádku i do /etc/fstab. Výhodou pvfstab je, že se dá jednoduše přenášet na různé klienty bez ohledu na to, jaká další zařízení budou mít připojena.



Obrázek 2.3: Architekura Gfarm FS (Zdroj: [18])

- *„Metadata cache server node“* – server cachující metadata. S existencí pouze jednoho centrálního metadata serveru je spojena řada problémů. Především soustředění velkého množství požadavků na jedno místo tvoří výkonnostní problém a obecně jakákoli centralizace je problematická s ohledem na náchylnost k poruchám hardwaru, či dočasnému přerušení síťového spojení. Toto, alespoň částečně, řeší Metadata cache server (démon *gfarmagent*). Jedná se o proxy server ležící mezi určitou částí clusteru a hlavním metaserverem, který na sebe přenáší část zátěže a požadavků filesystem nodů. Typické užití bývá, je-li celkový cluster složen z několika menších clusterů, například jednotlivých LAN sítí spojených do většího celku, tak v každé LAN je spuštěn právě jeden *gfarmagent* zastupující hlavní metaserver.

- *„Client node“* – Koncový uzel, kde se uživatelé připojují k Gfarmu. Klientská část umožňuje využití Gfarm více způsoby:

  1. Použitím Gfarm konzolových příkazů, nebo využitím nativního I/O API Gfarmu. K dispozici je množství command-line utilit jako *gfls* nebo *gfmkdir*, které jako obdoba standardních UNIXových příkazů, umožňují manipulaci s Gfarmem. Dále je možné využít API přeloženého do knihovny *libgfarm.a* pro přímý přístup z uživatelských aplikací.
  2. Využitím metody „syscall hook“, kdy se provede zachycení patřičných systémových volání z aplikací. K Gfarmu pak mohou aplikace přistupovat obvyklým způsobem, jako by FS byl namontován do systému souborů klientského počítače.
  3. Pomocí FUSE. Stejným způsobem jako GlusterFS je možné Gfarm exportovat do systému souborů klienta pomocí FUSE. Je k tomu ovšem třeba nainstalovat přídavek *GfarmFS-FUSE*, který není součástí balíku s Gfarmem.

Jako spojovacího rozhraní Gfarm využívá TCP/IP, nepodporuje spojení pomocí *infiniband*, či jiným způsobem. Další dle vývojářů unikátní vlastností je, že každý *filesystem node* je zároveň i klientem Gfarmu. Na *filesystem node* je tedy vždy možné přistupovat k celé instanci Gfarmu.

2.4.2 Instalace a konfigurace

Gfarm je možné zprovoznit obecně na jakémkoli OS Unixového typu, nejčastěji pak Linuxu nebo BSD systémech. Je jej možné přeložit a nainstalovat ze zdrojového archívu posloupností příkazů `./configure`, `make` a `make install` s případnými upřesňujícími parametry – zejména co se týče způsobu použití FS u klienta a výběru databázového backendu metaserveru apod. Nicméně tento způsob instalace je opět poněkud nepříjemný, pokud uvádíme Gfarm na větší cluster, proto jsou k dispozici RPM balíčky pro systémy, které je podporují.

Před uvedením do provozu je nejprve třeba Gfarm správně nakonfigurovat. K tomu slouží program *config-gfarm*, který je nutno spustit na metaserveru, jenž se musí nastavovat nejprve. Způsob použití *config-gfarm* je vypsán automaticky, je-li program spuštěn bez parametrů. Podrobně ho zde nebudu uvádět, je však nutno především zvolit databázový backend, způsob autentizace apod. Nakonfigurováním metaserveru vznikne soubor `gfarm.conf`, který je posléze třeba zkopírovat na ostatní nody. Pro konfiguraci metadata cache serveru se na patřičném počítači pokračuje obdobným konfiguračním programem *config-agent* a na počítačích zajišťujících běh *filesystem node* programem *config-gfsd*. Nachází-li se více součásti Gfarm na jednom počítači, není třeba `gfarm.conf` pro každou z nich kopírovat zvlášť, ale stačí jedna kopie na jednom stroji.

2.5 Další DFS

Čistě pro zajímavost uvádím krátký teoretický popis dalších dvou paralelních DFS, které sice byly původně také uvažovány, nakonec však nebyly zařazeny mezi testované systémy, jimiž se tato práce zabývá především.

2.5.1 dChache

DFS v mnoha ohledech odlišným je dCache[8]. Tento systém vznikl ve spolupráci několika institucí, z nichž jednou je Fermilab (Fermi National Accelerator Laboratory), která se podílí na výstavbě zatím největšího částicového akcelerátoru, známého jako Large Hadron Collider. Bude-li v tomto akcelerátoru spuštěn projekt, bude vyvinut tok vědeckých dat o rychlosti až 300 MB/s, který bude třeba uložit. K tomuto bude sloužit dCache. Jedná se o příklad jeho hlavního využití v budoucnu. V současnosti se již používá i v jiných souvislostech, hlavně tam, kde je třeba systému s jeho vlastnostmi.

Původní myšlenka, ze které vzešel název dCache, byla vytvořit mezistupeň mezi počítači, na nichž běží klientské aplikace, a mezi obsáhlým páskovým zařízením. Práce přímo s magnetickým páskovým záznamovým zařízením je poněkud nepraktická, dCache je proto clusterem většího počtu obvykle komoditních zařízení, která slouží jako cache umožňující transparentní a okamžitý přístup k datům na pásce. Toto bylo v průběhu vývoje rozvinuto až do dnešní podoby, kdy dCache může fungovat i jako samostatný síťový systém i bez dalšího záznamového zařízení, které by cachoval. Tím pádem se z něj stává i DFS ve smyslu dalších FS rozvíjených v této práci.

Jistou nevýhodou dCache je zajisté velice zmatečná a nejednotná dokumentace sestávající spíše ze slidů a textů použitých na nejrůznějších konferencích. I když má dCache poměrně slušným způsobem popsaný proces instalace a zprovoznění, informace o architektuře a detailnější charakteristiky jsou rozházené a nejsou snadno k nalezení.

Na rozdíl od předchozích DFS, které byly všechny psány v jazyce C, je dCache psán z větší části v Javě a není otevřený. DCache je free software, nicméně jeho zdrojové kódy

nejsou k dispozici. Hlavními součástmi dCache jsou server a klient. Server je aplikace, která běží na každém počítači, který tvoří cluster a zřejmě zároveň zpracovává i metadata, která ukládá do PostgreSQL databáze. Klient běží na počítačích, které provozují klientské aplikace a exportuje kompletní *namespace* dCache pomocí protokolu NFS2, čímž umožňuje použití POSIXových utilit pro práci se soubory. Klientská část také obsahuje API pro přímý přístup k dCache z uživatelských aplikací.

DCache je distribuován formou RPM balíčků použitelných na linuxových distribucích na RPM založených, kompatibilních s Red Hat.

2.5.2 MogileFS

Druhým netestovaným DFS je MogileFS[12]. Jedná se o systém vyvíjený společností Danga Interactive, který se svými vlastnostmi a stavbou řadí mezi paralelní DFS s chybovou tolerancí.

MogileFS je napsaný v Perlu, což, jakožto skriptovací jazyk je zajisté poněkud netypický pro implementaci FS, ale je zajímavý pro implementaci DFS. Součástí MogileFS je tracker – *mogilefsd* proces, který koordinuje a zajišťuje chod celého systému. Autoři doporučují použití 2 trackerů ve vzájemné spolupráci, aby byl odstraněn "single point of failure". Vlastní cluster je tvořen sítí WebDAV *http* serverů, tedy k vlastnímu přenosu dat se používá protokol *http*. Pro tento účel autoři napsali server komponentu - *mogstored*. Z uživatelských aplikací je potřeba přistupovat pomocí speciálního API. Podpora FUSE je již rozpracována, nicméně není dokončena.

Kapitola 3

Experimentální činnost

Tato kapitola rozvádí testování systémů obecně popsaných v předchozí kapitole, a analyzuje získané výsledky testů. Začíná popisem testovací sítě, pomocných skriptů a návrhem prováděných testů. Pokračuje podkapitolami, které se věnují jednotlivým distribuovaným FS. Každá z těchto podkapitol obsahuje informace o instalaci a konfiguraci DFS na testovacím clusteru, a analýzu výsledků testů.

3.1 Testovací síť

K práci s vybranými DFS nám byly fakultou poskytnuty počítače a další hardware, což umožnilo sestavit dostatečně velký cluster, který jsem použil pro testování všech DFS, dále rozebíraných v této kapitole.

Cluster sestával z devíti stejných počítačů s 64 bitovým procesorem Intel Pentium 4 Core2Duo 2.6Ghz, 2 GB RAM, a 250 GB pevným diskem. Podstaná byla přítomnost gigabitové síťové karty netvořící rychlostní omezení, jež se tak nejčastěji přesunulo na pevný disk. Síťové karty byly spojeny výkonným gigabitovým přepínačem. Z počítačů jsme se rozhodli použít osm na vytvoření samotného testovacího clusteru, zatímco jeden sloužil jako nápomocný stroj pro nejrůznější úkoly potřebné pro administraci clusteru. Především fungoval jako NFS server poskytující společný diskový prostor pro všechny počítače clusteru. Tento prostor byl připojen na všech počítačích na stejném místě, což nám ušetřilo práci s kopírováním nejrůznějších konfiguračních souborů některých DFS. Mimo jiné v něm byly umístěny testovací skripty, a tak grafy, které naše testy generovaly, byly ukládány na jednom místě.

Na všechny počítače testovacího clusteru jsme nainstalovali CentOS Linux 5.1, protože tento systém je v současné době rozšířen i ve škole a v praxi by tedy některý z DFS běžel na něm. Instalovali jsme pouze potřebný software, aby byly splněny závislosti všech DFS. Na devátý, obslužný počítač (dále Server), jsme provedli instalaci komplexní, včetně např X11 pro lepší správu clusteru – což nakonec nebylo třeba, protože jsme v praxi s celým clusterem pracovali přes SSH vzdálený login z našich notebooků, a to i se Serverem.

3.2 Návrh testů

Náš testovací cluster byl sestaven z počítačů s poměrně slušným hardwarem. Nicméně 8 počítačů neumožňovalo otestovat všechny výkonnostní aspekty testovaných distribuovaných FS. Zvláště při praktickém použití ve velkých clusterech s řádově desítkami počítačů,

případně propojením pomocí infiniband, vznikají nové, odlišné podmínky, které bychom mohli jen stěží napodobit.

Nicméně, jak již bylo zmíněno, naše snažení bylo uzpůsobeno tomu, aby přineslo informace o DFS především v rozměrech, ve kterých by mohly být použity na naší škole. Naše testy tedy sice přináší i poměrně přesné informace o výkonu na této menší konfiguraci, ale dělat z nich hlubší závěry o výkonnostních poměrech různých implementací paralelních distribuovaných FS nebude vhodné. Proto jsem se zaměřil spíše na pochopení dění v DFS během testů. Dal-li test určitý výsledek, tak cílem nebylo ani tak zjistit přesnou např. rychlost, ale spíše proč je rychlost taková, tedy která hardwarová součást, nebo vnitřní pochod DFS byl tím, který se na výsledku nejvíce podílel.

Následuje přehled uskutečněných testů:

- Zápis – zápis 1 GB náhodných dat
- Čtení – čtení 1 GB souboru
- Opakovaný zápis – zápis 1000 x 1 MB náhodných dat
- Opakované čtení – čtení 1000 x 1 MB souboru
- Zápis na konec souboru – zápis na konec souboru několika klienty zároveň, test souvislosti souboru
- Test výpadku dataserveru během I/O operace

Čtením a zápisem jsme testovali hrubý výkon vybraných DFS, a jak se ukázalo, významně ovlivněný cachem. Původním záměrem bylo vliv cache eliminovat tím, že se nejprve bude náhodnými daty zapisovat soubor, který se pak bude vzápětí číst. Jak se však ukázalo, do I/O operací byla intenzivně zapojena i cache na dataserverech, ve které zůstával taktéž i zapisovaný soubor.

Testy opakovaného čtení a zápisu byly zaměřeny na rozpoznání vlivu metaserveru na výkon DFS. Přenášelo se sice celkově stejné množství dat, nicméně po menších částech. Před zápisem každého dalšího MB byl vždy soubor uzavřen a znovu otevřen.

Testem zápisu na konec souboru jsme u vybraných DFS netestovali výkon, který jak jsme předpokládali, by byl obdobný jako u obyčejného zápisu. Místo toho jsme testovali schopnost dodržet jednu z vlastností POSIX C I/O knihovny – pokud bude více souborů zapisovat do jednoho souboru zároveň, a budou ho mít otevřený jako *append*, data se nebudou navzájem přepisovat. Taktéž jsme zjišťovali, zda budou data zapsaná více klienty uložena za sebou, nebo budou proložena.

3.3 Testovací skripty

Před uskutečněním testů jsme si kladli otázku, jakým způsobem je nejlépe provést. Bylo potřeba vzít v potaz několik faktorů.

- Velikost clusteru, tedy počet aktuálně zapojených prvků DFS
- Synchronizaci testů v situacích, kdy museli připojení klienti provádět vybranou I/O operaci ve stejné chvíli
- Způsob zadání a vlastního spuštění testů

Za tímto účelem jsme napsali dva programy komunikující spolu pomocí *tcp/ip* a *BSD sockets* klasickou metodou klient/server. Jako implementační jazyk jsme zvolili Python. Ten jsme vybrali proto, že jde o jazyk interpretovaný – nebylo nutné kompilovat při každé, během testování časté, změně kódu. Dále se jedná o jazyk vysokoúrovňový, dostatečně jednoduchý a přitom nabízející dobrou nízkoúrovňovou práci s *BSD sockets* a I/O operacemi.

První skript pojmenovaný „dfsSingleton“ jsme spouštěli jako klienta pouze na Serveru. Spouštěl se zvlášť pro každý prováděný test. V tomto skriptu bylo nastavováno, které počítače hrály jakou roli podle momentálně zvolené konfigurace testovaného DFS. Dále zde byly zapsány cesty k souborům, zpravidla v místě připojení zkoumaného FS, se kterými měly jednotlivé počítače pracovat při vykonávání testů.

Ve skriptu pojmenovaném „dfsServer“ pak byly implementovány vlastní testy. Skript běžel na všech počítačích clusteru, naslouchal na vybraném portu a plnil požadavky *dfsSingletonu*, dokud nebyl explicitně ukončen.

Celý postup při vykonání jednoho testu tedy byl takový:

1. Editace předpřipravených seznamů hostnames počítačů – nastavení účastníků testu. Přitom se rozlišovaly počítače, na kterých běželi klienti DFS aktivně vytvářející požadované I/O operace a počítače se spuštěnými pouze datovými resp. metaservery, které test neřídily, ale přesto bylo třeba po jeho ukončení sebrat výsledky sledování hardwaru a vygenerovat z nich grafy (viz dále).

2. Uživatel zadává název testu jako parametr *dfsSingletonu* a spouští jej.

3. *DfsSingleton* odesílá zadání testu a cesty k potřebným souborům všem naslouchajícím procesům *dfsServer* a začíná čekat na příchod výsledků.

4. *DfsServer* přijímá zadání a spouští patřičný test, popřípadě čeká na informaci o ukončení testu, běží-li na neklientské části zkoumaného FS.

5. Testy jsou dokončeny. Jednotlivé *dfsServery* spouští shellový skript, který zajistí generování grafů z údajů zaznamenaných nástrojem popsaným v následující sekci.

6. *DfsServer* vrací informace o průběhu a délce testu čekajícímu *dfsSingletonu*.

7. *DfsSingleton* vypisuje výsledky testů a končí.

3.3.1 Statistiky

O sběr statistických dat a tvorbu grafů se zasloužil kolega Martin Tomec. Následuje tedy jeho popis[16].

Během jednotlivých testů bylo nutné zaznamenávat statistiky využití sítě, paměti, disků a procesoru pro pozdější analýzu. Na zaznamenávání těchto statistik jsme použili nástroj HotSaNIC [1]. Tento nástroj zaznamenává (nejen) výše uvedené statistiky pomocí perlových skriptů a ukládá je ve formátu rrd. Nevýhodou tohoto nástroje je, že celá implementace počítá s pevně danou vzorkovací periodou 10 s. Ta je samozřejmě na některé testy příliš dlouhá, takže ji bylo nutné upravit na několika místech ve zdrojových souborech. Otázkou je, zda častější spouštění skriptů neovlivní výsledky testů, ale i když jsme zvolili vzorkování jednou za sekundu, skripty nevyužili procesor na více než 2%. Jediné, čím mohly

[1] Jako alternativní nástroj jsme zvažovali použití MRTG. Ten již také podporuje zaznamenávání do formátu rrd, ale v době testování nepodporoval nastavení libovolné periody záznamu. V aktuální verzi je podle dokumentace už možné nastavit periodu záznamu i v sekundách

ovlivnit výsledky testů, byly přístupy na disk. Navíc by se tato chyba objevila u všech testovaných systémů a neovlivnila by tedy jejich srovnání. Statistiky byly průběžně zaznamenávány na lokální disk a po skončení testu zkopírovány na server. Formát rrd totiž uchovává (v nejvyšším rozlišení) pouze několik posledních záznamů. Starší záznamy sumarizuje, aby soubor se záznamy zůstal rozumně velký. Výchozí nastavení nástroje HotSaNIC při zaznamenávání po jedné sekundě uchovává posledních 12 minut s rozlišením 1 s, což je dostatečné pro většinu testů.

Drobným zklamáním u tohoto formátu pro mě byla jeho platformová závislost. Z údajů zaznamenaných na jednom systému nelze například generovat grafy na druhém systému.

Ze všech statistik, které nástroj HotSaNIC umožňuje zaznamenávat, byly vybrány pouze následující:

- Procesor (jednotlivá jádra i celkové vytížení)
- Paměť RAM
- Síťové rozhraní
- Rychlost čtení z disku a zápisu na disk (včetně počtu přístupů)

3.4 GlusterFS

Prvním distribuovaným FS, který jsem testoval na našem clusteru, byl GlusterFS. Obsahem této podkapitoly je popis mých zkušeností s jeho instalací, konfigurací a zamyšlení nad výsledky testů.

Instalace

Vzhledem k tomu, že na našem clusteru běžel Red Hat EL5 kompatibilní s CentOS 5.1, zvolil jsem instalaci pomocí dodávaných RPM balíčků. Rychlá instalace předkompilovaných RPM balíčků též ušetřila čas, který by jinak mohl být nutný pro kompilování na každém počítači clusteru. GlusterFS je dodávaný ve čtyřech balících obsahujících jednotlivé součásti. Není tedy nutné instalovat klienta *glusterfs* a s ním spojené FUSE i tam, kde měl běžet pouze server *glusterfsd*. Nicméně v plánu bylo provést testy na nejrůznějších konfiguracích GlusterFS, a proto jsem všude instaloval kompletní sadu balíčků[4] kromě *devel*, a to tyto:

- `glusterfs-server-1.3.7-1.x86_64.rpm` - server *glusterfs*
- `glusterfs-client-1.3.7-1.x86_64.rpm` - klient *glusterfsd*
- `glusterfs-common-1.3.7-1.x86_64.rpm` - především společné *translators*

Balíčky a tedy i testovaný DFS byly verze 1.3.7. [2]

Nejen pro tuto instalaci jsme si vytvořili pomocné skripty pro vzdálené spouštění příkazů *rrSingleton* [3] a *rrServer* založené na kódu výše popsaných testovacích skriptů. Na Serveru jsme pak pomocí *rrSingletonu* spouštěli příkazy, které se vykonávaly na ostatních počítačích, kde běžel *rrServer*. To nám umožnilo jednodušší administraci clusteru a konfigurace testovaných DFS.

[2] V době psaní této práce se již objevila verze 1.3.8, která je v tuto chvíli dodávána pouze v jednom balíku *glusterfs-1.3.8-1* (opět nepočítám-li *devel*)

[3] „rr“ znamená **r**emote **r**un

Konfigurace

Jak bylo uvedeno u obecného popisu, GlusterFS se konfiguruje s pomocí souborů označovaných jako „Volume Specification“ (dále VS). V konkrétní instanci GlusterFS pak má každý *glusterfsd* svůj vlastní VS, lišící se obsahem podle nastavení serveru. Pro klienty *glusterfs* existuje jeden společný VS, v němž dochází ke sjednocení všech serverů do clusteru pomocí *translatoru cluster/unify*.

VS má přesně definovaná syntaktická i sémantická pravidla. Sestává z bloků `volume`, které mají vždy jako `type` uvedeny název *translatoru*, který je implementuje. Následují další nastavení uvozená slovem `option` s možnostmi podle právě používaného *translatoru*. Bloky volume se píšou za sebe a bloky některých *translatorů* mohou logicky spojovat jiné bloky.

Uvedu příklad konfiguračního souboru klientů, který byl použit v jedné z konfigurací[4] během testování. Příklad konfiguračního souboru dataserveru najdete na přiloženém CD.

```
volume client1
  type protocol/client
  option transport-type tcp/client
  option remote-host s1
  option remote-port 6996
  option remote-subvolume brick
end-volume
```

`Volume client1` specifikuje náležitosti jednoho ze vzdálených *glusterfsd* serverů. Následují, pro podobnost zde neuvedené, bloky `client2` a `client3`, které specifikují další dva servery v tomto rozložení clusteru.

```
volume bricks
  type cluster/unify
  subvolumes client1 client2 client3
  option scheduler alu
  option alu.order disk-usage:read-usage:write-usage:open-files-usage
end-volume
```

Dalším je blok *translatoru cluster/unify*, který spojuje tři definované vzdálené *glusterfsd* do jednoho logického *namespace*. Tímto způsobem je možné kombinovat funkce nejrůznějších *translatorů* a vytvářet zajímavé a velice specifické konfigurace [5]. Dalším důležitým nastavením v rámci *cluster/unify* je výběr tzv. scheduleru, tedy plánovače, který má na starost přerozdělování nových souborů mezi dataservery. Zvlášť proto, že GlusterFS nemá metaserver, a tedy ve většině[5] konfigurací nedělí soubory na menší části, které se nacházejí na různých serverech, je správným výběrem scheduleru možné optimalizovat činnost GlusterFS na vybranou konkrétní aplikaci.

V rámci našeho testování jsem používal plánovač *Adaptive Least Usage*, tzv. ALU. Dle autorů [5] je nejvyspělejší a nejvhodnější pro obecné použití. Jak je patrné z předchozí ukázky, je možné nastavit pořadí faktorů, na které se bere ohled při výběru dataserveru pro uložení nového souboru.

[4] Jednalo se o konfiguraci složenou ze 3 dataserverů a libovolného množství klientů

[5] Pro GlusterFS je implementován i translator cluster/stripe. Autory však jeho použití, vzhledem k celkové myšlence GlusterFS, neni mimo několika specifických případů užití doporučováno. Viz FAQ v dokumentaci[7]

Existují i další schedulery podrobně popsané na GlusterFS WIKI [7]. Zajímavý je zejména scheduler *switch*, pomocí kterého je možné přerozdělovat umístění souborů podle jejich typu na vybrané servery. Zkombinuje-li se použití tohoto plánovače a *translatoru* AFR pro vytváření zrcadlových kopií vybraných serverů, je možné vytvořit systém, ve kterém se například soubory *.dat replikují a uchovávají ve více kopiích, zatímco ostatní soubory takto zálohovány nejsou.

3.4.1 Prováděné testy

GlusterFS jsem podrobil výše uvedené skupině testů na několika konfiguracích. Testoval jsem konfigurace s jedním až třemi dataservery a vždy s několika různými počty připojených klientů. Každý test jsem realizoval několikrát, aby bylo dosaženo statisticky lepších výsledků, popřípadě zamezeno zkreslení výsledků náhodným výkyvem v době testu.

3.4.2 Analýza výsledků

Test	1 klient	2 klienti	4 klienti
Zápis	16,5	x	x
Čtení	11,3	19,3	154
Op. zápis	17,5	x	x
Op. čtení	< 1	< 1	< 1

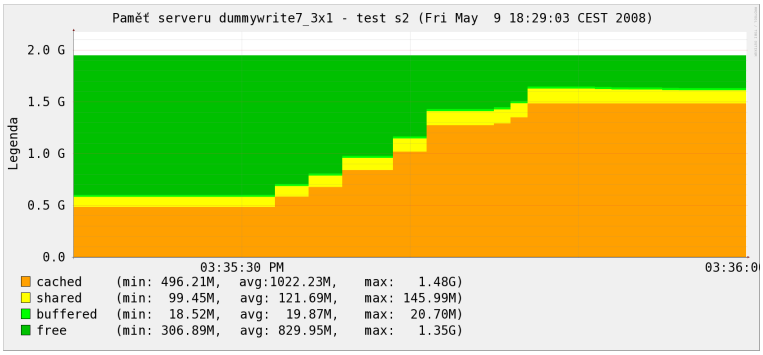
Tabulka 3.1: Trvání testů GlusterFS – uvedeno v sekundách

Jak již jsem se zmínil, GlusterFS nerozděluje soubory na části, ale každý nově vzniklý soubor celý uloží na jeden konkrétní dataserver. K výběru vhodného serveru slouží jeden ze schedulerů, který dle svého nastavení dataserver zvolí. Proto jsou v tabulce zaznamenány výsledky pouze pro různé počty klientů, nikoli dataserverů – některé testy pracovaly právě s jedním souborem, a tím pádem prakticky i jen s jedním dataserverem, na kterém byl dotyčný soubor právě uložen. Jiné testy pak používaly několik souborů různě rozložených v clusteru. V tabulce jsou uvedeny časy naměřené při práci jen s jedním dataserverem. Rozdíly výkonu při používání více souborů během testování i zamyšlení se nad vlivem takovéto práce DFS na praktické použití bude součástí následujícího textu.

Zápis

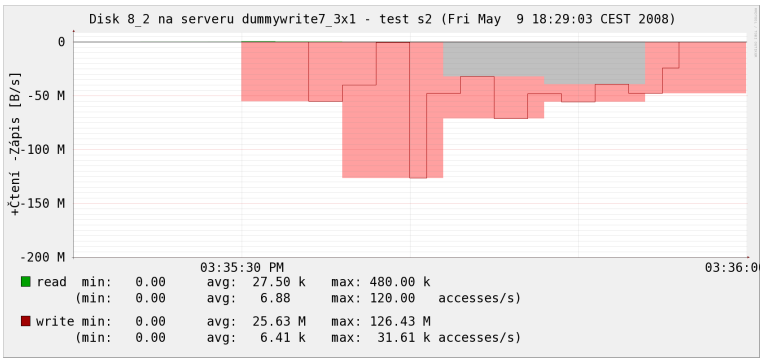
Při testu zápisu vygeneroval skript *dfsServer* běžící na vybraném klientovi náhodná data, která posléze zapisoval po kilobytových blocích do souboru nově vytvořeného v oblasti vyexportovaného GlusterFS. Tak vznikl 1 GB soubor. Sítí se tedy od klienta k jednomu nebo více dataserverům musel tento objem dat přenést.

Pokud se testu účastnil jeden klient, přenášel data na dataserver určený schedulerem k uložení souboru vyšší rychlostí, než byla maximální rychlost zápisu disku dataserveru. Díky skutečnosti, že operační paměť dataserveru měla kapacitu 2 GB, zatímco soubor velikost pouze 1 GB mohl být do cache nahrán celý soubor. Na disk serveru se data mezitím zapisovala rychlostí, která byla přibližně o 15 MB/s nižší. Situaci dokumentují grafy 3.1 a 3.2. Graf 3.1 zobrazuje obsazení paměti dataserveru během testu.



Obrázek 3.1: Obsazení paměti dataserveru během zápisu jedním klientem

Zápis na disk začal ve stejné chvíli, jako se spustil přenos dat sítí. Disk četl data z cache a zapisoval nižší rychlostí, než do ní data přicházela. Proto disk zapisoval ještě přibližně dalších 10 sekund po dokončení přenosu dat do cache.



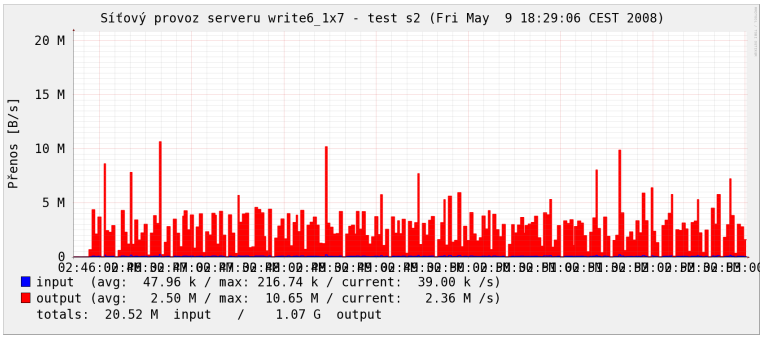
Obrázek 3.2: Činnost disku dataserveru během zápisu jedním klientem

Z pohledu klienta tedy zápis skočil o přibližně 10 sekund dříve, než tomu tak skutečně bylo. Nabízí se otázka, byla-li paměť dostatečně veliká, proč se data do cache dataserveru nezapisovala plnou síťovou rychlostí. Je možné, že tomu bylo proto, že ve stejnou chvíli, kdy do cache data přicházela, byla zároveň čtena diskem, čímž mohla cache být částečně brzděna. Na druhou stranu, při rychlostech dnešních pamětí RAM, se zdá, že by k tomu docházet nemělo. Domnívám se tedy, že toto zbrzdění je výsledkem použité implementace cache.

Ve všech našich testech měla cache skutečně velký význam. Téměř vždy urychlila I/O operaci – pravděpodobně ke spokojenosti klienta v praktické aplikaci. Abych vyzkoušel rychlost zápisu s pokud možno alespoň částečně eliminovanou cache, nechal jsem zapsat 10 GB soubor, který byl tedy pětkrát větší než kapacita paměti RAM. Průměrná rychlost zápisu byla 46 MB/s, což je o 16 MB/s méně než v případě zápisu 1 GB souboru.

Test opakovaného zápisu byl zaměřený především na DFS obsahující metaserver, tedy komponentu, kterou je třeba kontaktovat při pokusu o otevření/zavření souboru - vzniká aspekt, ktery může ovlivnit přenos dat. GlusterFS ovšem metaserver nemá, opakovaný zápis 1000 x 1 MB soubor tedy probíhal stejným způsobem i stejně rychle jako přesnost 1 GB souboru.

Současný zápis více klientů do jednoho souboru pochopitelně není možný (proto prázdná



Obrázek 3.3: Činnost sítě dataserveru během zápisu sedmi klienty

místa v tabulce 3.1). Zapisovalo-li více klientů do svých vlastních souborů, záleželo především na tom, na kterém dataserveru byly tyto soubory umístěny. Jestliže docházelo k zápisu do dvou souborů na jednom dataserveru dvěma klienty, rychlost zápisu každého z nich klesla přibližně na polovinu. S větším množstvím klientů klesala průměrná rychlost dokonce více než lineárně. Další zpomalení pak bylo způsobeno narůstajícím počtem požadavků na zápis na disk, který musel zapisovat naráz na více míst, čímž se začala projevovat doba vybavení a doba vystavení hlav.

Tuto situaci dokumentuje graf 3.3 NUM. Jedná se o současný zápis sedmi klientů, přičemž všechny soubory jsou umístěny na jednom dataserveru. Rychlost přenosu dat od libovolného z klientů klesla na průměrných 2,5 MB/s, což není sedmkrát, ale dvacetpětkrát pomalejší.

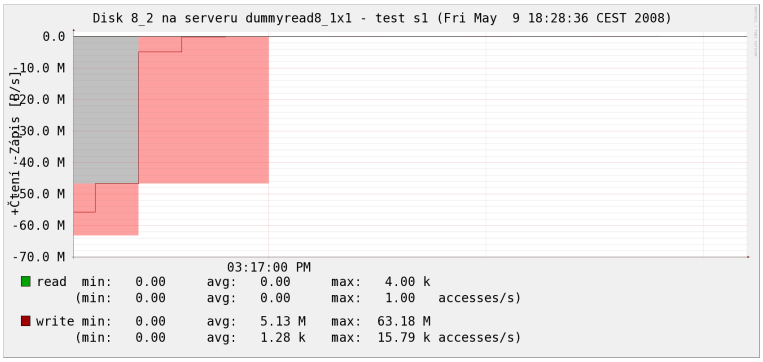
Zdá se, jako by GlusterFS nebyl příliš efektivní. Nicméně nezapomínejme na scheduler, který soubory inteligentně (nebo jinak, viz *switch*) rozděluje na různé servery. Pokud tedy nemáme aplikaci, která pracuje stále s jedním souborem, je pravděpodobné, že zatížení GlusterFS bude poměrně dobře rozděleno. A to zvláště v případech, kdy na FS současně přistupuje množství různých klientů pracujících s různými soubory.

Když jsem připravil test, ve kterém tři klienti zapisovali do souborů umístěných na různých dataserverech, rychlost přenosu dat od každého z klientů byla stejná jako v testech, kdy klient v jednom momentě pracoval s GlusterFS sám.

Čtení

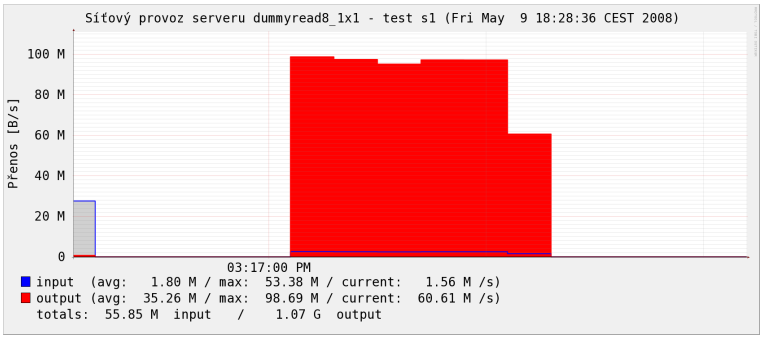
Pro čtení z exportovaného GlusterFS platila stejná pravidla rozdělění souborů a výkonu jako u zápisu. Čtený soubor byl vždy celý uložen na jednom dataserveru. Jestliže k němu přistupovalo v jednu chvíli více klientů, museli se dělit především o síťové zdroje. Přistupovalo-li v jednom testu více klientů k více souborům a na více serverech, navzájem se neovlivňovali a měli k dispozici všechny zdroje daného serveru. V opačném případě bylo čtení brzděno obdobným způsobem jako u zápisu – v tabulce 3.1 je to vidět na rozdílných časech testů čtení dvěma a čtyřmi klienty.

Více než zápis bylo cachem ovlivněno čtení, opět hlavně na straně dataserverů. Významné pro rychlost čtení bylo, zda se již celý soubor nebo jeho část nacházela v paměti – například po předchozím čtení nebo zápisu. Grafy 3.4 a 3.5 znázorňují typické, cache ovlivněné čtení. V následujícím grafu je zobrazeno I/O disku dataserveru, na kterém se nacházel čtený soubor. Nejprve je vidět končící zápis souboru, který se ihned v dalším testu četl.



Obrázek 3.4: Činnost disku při čtení s využitím cache

Jak ukazuje obrázek 3.5, ihned po dokončení zápisu začalo čtení, které však již nemělo na chod disku vliv, protože celý soubor byl v cache, ze které byl sítí posílán klientu.



Obrázek 3.5: Činnost síťě při čtení s využitím cache

Rychlost v tomto testu (viz tabulka 3.1) byla omezena výhradně maximální přenosovou rychlostí gigabitové sítě, čemuž hodnoty okolo 100 MB/s odpovídají.

Pokud soubor nebyl v cache, musel být čten přímo z disku a rychlost čtení tedy odpovídala maximální rychlosti disku.

Zajímavě GlusterFS využil cache během testování čtení jednoho souboru vícero klienty. V situaci, kdy četli soubor 2 klienti, první klient přistupoval k souboru o něco později než druhý. První klient svým čtením vytvářel cache, ze které mohl vzápětí číst i druhý klient, aniž by musel jakkoli zatěžovat pevný disk. Při následném přenosu byla pak celková rychlost výstupu sítě dataserveru využita dvojnásobně oproti maximální rychlosti čtení disku – jak každý klient přesouval data rychlostí, jakou je měl k dispozici. Rychlost čtení tak v této situaci byla pro každého z klientů stejná jako v případě, kdy by k souboru přistupoval pouze sám. Shodou okolností byla maximální propustnost sítě zhruba dvojnásobná jako rychlost disku v tomto testu. Kdyby četli více než 2 klienti zároveň, rychlost přijmu dat klientem by již byla menší.

Cache měla zásadní vliv i na opakované čtení. Tento test četl tisíckrát 1 MB soubor, čímž přečetl celkem 1 GB dat. Vzhledem k neexistenci metaserveru nebyla nutná žádná síťová komunikace mimo samotného přenášení dat. Cache se pro změnu uplatnila na straně klienta. Klient přečetl soubor pouze jednou, uložil si ho do své cache, a když mu přišel pokyn na opětovné čtení tohoto souboru, tak ho získal lokálně. Celý test tak trval méně než 1 sekundu, nehledě na to, kolik klientů se ho účastnilo.

Zápis na konec souboru

Pokud do jednoho souboru zapisovalo více klientů ve stejnou chvíli 1 MB, data byla sice uložena na přeskáčku, ale k žádné ztrátě nedošlo. V extrémnějším případě, kdy tři klienti připisovali celkem 1 GB po 1 MB blocích, skončil zápis vytvořením pouze 2 GB souboru. Došlo tedy ke ztrátě třetiny zapisovaných dat. GlusterFS tudíž bezvýhradně toto POSIXové pravidlo nesplňuje.

Výpadek

Chování GlusterFS během výpadku jednoho z dataserverů jsem testoval tak, že během kopírování souboru jsem vybranému serveru odpojil síťový kabel. Kopírování se pozastavilo do doby, dokud jsem jej opět nepřipojil. Poté se znovu rozeběhlo a celý soubor byl správně přenesen. Pokud jsem vypojil kabel dataserveru, na kterém nebyl kopírovaný soubor uložen, nebo na který se právě nekopíroval, nebylo kopírování nijak narušeno.

Během tohoto testu nebyl dotyčný dataserver zrcadlován s pomocí *translatoru cluster/afr*. Tento *translator* pracuje tím způsobem, že vždy obsah celého jednoho dataserveru zrcadlí na jeden nebo více dalších serverů. Vytváří tedy podobný způsob zálohy jako poskytuje diskové RAID pole. Výhodou je, že dojde-li k odpojení celého počítače, jsou data stále přístupná[5], zatímco pokud by bylo použito k zálohování RAID pole, budou dostupná pouze, jestliže dojde k výpadku disku. Toto řešení je zajisté nákladnější, protože je třeba pořídit celý další počítač.

3.5 Parallel Virtual File System 2

Dalším testovaným systémem byl Parallel Virtual File System 2.

Instalace

Protože autoři PVFS nedávají k dispozici předkompilované balíčky, instalaci[15] jsem provedl ze zdrojového archívu. Většina ze závislostí PVFS je součástí téměř každé linuxové distribuce[6], ostatní bylo třeba doinstalovat:

- Berkeley DB verze 3 nebo 4
- pthreads

Během samotné kompilace jsem postupoval stejným způsobem jako je popsáno v druhé kapitole - posloupností příkazů `./configure`[7] , `make` a `make install`. Na počítačích, které měly sloužit jako klientské, bylo navíc třeba spustit `make kmod`, čímž došlo ke kompilaci modulu jádra.

Překládaný zdrojový kód se nacházel na NFS, které bylo připojeno ve všech počítačích. Vhledem k tomu, že konfigurace všech počítačů clusteru a jejich softwarové vybavení bylo prakticky totožné, stačilo provést překlad pouze na jednom počítači. S využitím našeho skriptu pro vzdálené spouštění jsem pak vykonal `make install` na ostatních počítačích, čímž se PVFS nainstalovalo v celém clusteru.

[6] Podpora Asynchronous I/O (AIO), gcc >= 2.96 a GNU Make

[7] Na počítačích, kde jsem měl v úmyslu provozovat pvfs2-client, jsem zde volbou *-with-kernel=/cesta/ke/hlavičkám/kernelu* musel uvést cestu ke zdrojovému kódu běžícího kernelu

Konfigurace

Vždy, když jsem měnil počet *pvfs2-server* fungujících jako dataservery nebo metaservery, tedy měnil konfiguraci clusteru, bylo třeba spustit skript *pvfs2-genconfig*. Tomuto skriptu jsem interaktivní formou zadal potřebné údaje, zejména *hostnames* počítačů, na nichž měly *pvfs2-server* fungovat jako dataservery a také na těch, které měly zastávat roli metaserveru. *Pvfs2-genconfig* vytvořil konfigurační soubor `/etc/pvfs2-fs.conf`, který je standardně třeba rozkopírovat na všechny počítače účastnící se clusteru. Struktura tohoto souboru je podobná XML a není těžké ho v případě potřeby ručně editovat. *Pvfs2-fs.conf* je příliš dlouhý na to, abych sem vkládal jeho ukázku, proto příklady těchto souborů použité při různých konfiguracích během testování naleznete na přiloženém CD.

3.5.1 Prováděné testy

Na PVFS jsem provedl stejnou skupinu testů jako u GlusterFS.

3.5.2 Analýza výsledků

V tabulce 3.2 jsou uvedeny výsledky testů PVFS. PVFS má architekturu významně odlišnou od GlusterFS. Všechny soubory uložené v tomto DFS jsou děleny na části, které jsou uloženy na počítačích s různými běžícími *pvfs2-server*. [8]. Proto jsou v tabulce uvedeny výsledky testů všech zkoumaných konfigurací. Hodnoty jsou doby trvání testu ze strany jednoho klienta v sekundách. Všechny testy opět přenášely ať směrem ke klientovi, ať směrem od něj vždy 1 GB. V testech opakovaného čtení, resp. zápisu, pak přenesly 1024 x 1 MB.

Informace typu „2 x 4“ znamená, že daná konfigurace sestávala ze dvou dataserverů a testu se účastnili 4 klienti. Ve všech testech mimo „4 x 4“ byl v clusteru přítomný vždy jeden metaserver. V konfiguraci „4 x 4“ jsem zkoušel provádět testy s jedním i dvěma aktivními metaservery, a jak se ukázalo, na cluster naší velikosti to nemělo vliv.

U některých testů, jichž se účastnilo více klientů, měly někdy testy pro tyto různé klienty odlišnou dobu trvání. V tabulce 3.2 je vždy uveden průměr těchto časů, chceme-li proto získat celkovou propustnost sítě v MB/s, stačí vydělit číslo 1024 touto hodnotou a vynásobit počtem klientů.

Test	1 x 1	1 x 2	2 x 1	2 x 2	2 x 3	2 x 4	4 x 4
Zápis	11,6	27	11,1	19,7	23	28	22
Čtení	11,4	25	11,0	15,1	17,2	50	24
Op. zápis	16,6	x	15,4	24	x	34	27
Op. čtení	16,3	26	15,9	x	x	x	30

Tabulka 3.2: Testy PVFS 2 – uvedeno v sekundách

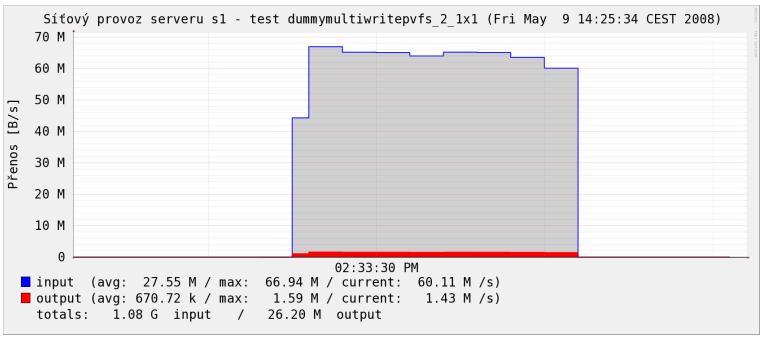
Čtení a zápis

PVFS vždy soubory rozdělené na části – *stripes* ukládal na dataservery víceméně rovnoměrně. Vždy, když jsem počítal velikost dat, které na dataserverech zabírala data uložená PVFS, byly tyto velikosti přibližně stejné. Prakticky u každého testu, jedno jestli čtení nebo

[8] Dále je budu nazývat *dataservery* a zastává-li pvfs2-server funkci metaserveru – tak metaserver(y). Označím-li počítač jako klient, myslím tím počítač se spuštěným *pvfs-client*

zápisu, byla proto zátěž rozdělena víceméně rovnoměrně. To se vyplatilo zvláště v případech, kdy více klientů přistupovalo k jednomu souboru ve stejné chvíli. Vzhledem k tomu, že byly všechny soubory rozmělněny, pak v praxi ani příliš nezáleželo na tom, zda každý klient pracoval s vlastním souborem, anebo (v případě čtení) jej sdílel s jinými.

Cache měla opět na výsledek testů velký vliv. Nápadné je to zvláště tehdy, pokud četlo menší množství klientů. Rychlost čtení 1 GB, četl-li jeden klient, je prakticky stejná jako tomu bylo u GlusterFS. Docházelo zde ke stejně efektivnímu využití cache. Byl-li soubor již dostupný například z předchozího zápisu, tak se četl z cache a využíval plného potenciálu sítě.



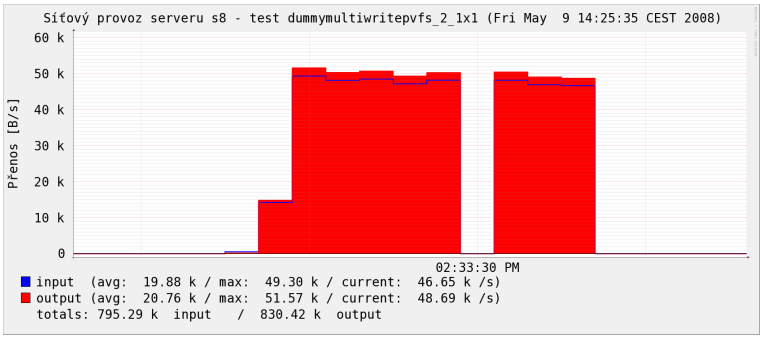
Obrázek 3.6: Příjem dat dataserverem zpomalovaným komunikací s metaserverem

Nicméně, pokud se zapisovalo, dovedl cache PVFS využít lépe. Celý soubor byl opět nahrán do cache dataserveru vyšší rychlostí, nežli je maximální zápis disku. PVFS však dokázal v tomto případě využít maximálního potenciálu sítě, a tak dosáhl stejné rychlosti zápisu jako u plně cachovaného čtení. Výhodné je, že při zápisu bude takto cache využito vždy (postačí-li RAM). Plně cachované čtení je přece jenom spíše ideálním I/O dějem a v praxi bude zřejmě většinou cachováno jen částečně.

Jako ukázka nejméně ideální situace slouží test konfigurace „2 x 4“, kdy se doba potřebná pro čtení každým ze čtyř klientů vyšplhala na průměrných 50 sekund. Každý z klientů četl svůj vlastní soubor, který nebyl ani částečně v cache. Proti ostatním testům (kdy bylo ze čteného souboru zpravidla v cache tolik, kolik bylo možné), se jedná o extrém, který jsem přivodil restartem obou dataserverů.

Sledujeme-li hodnoty v tabulce 3.2, dochází pochopitelně k jejich zvyšování s měnícím se počtem klientů. Hodnoty zpravidla přibližně odpovídají nutnému rozdělení prostředků, k jakému musí docházet v dané konfiguraci. Je vidět, že doba potřebná k přenosu dat se zvedá poněkud rychleji než lineárně ve vztahu k množství dataserverů a klientů. Dokumentuje to například výsledek testu konfigurace „4 x 4“, kdy doba není stejná jako v konfiguraci „1 x 1“, ale prakticky dvojnásobná. To je způsobeno skutečností, že všichni klienti komunikovali se všemi servery, množství propletené komunikace se tedy muselo projevit.

Testy opakovaného zápisu byly provedeny především kvůli možnému vlivu metaserveru na výkon - jeden soubor bude opakovaně otvírán, znovu zapisován nebo čtený, a poté zavírán. A skutečně, protože PVFS má metaserver, zápis i čtení tímto způsobem vyvolalo potřebu stálé komunikace s ním. Prakticky ve všech konfiguracích trval test průměrně o 30 % delší dobu než v testech s 1 GB souborem. Graf 3.6 znázorňuje tok dat přijímaných dataserverem od klienta brzděných drobnou komunikací, která probíhala ve stejnou dobu s metaserverem. Jeho síťové I/O je zobrazen v grafu 3.7. Provedl jsem i obdobu tohoto testu, kdy zůstal přepisovaný soubor po celou dobu otevřený, a po ukončení zápisu každého

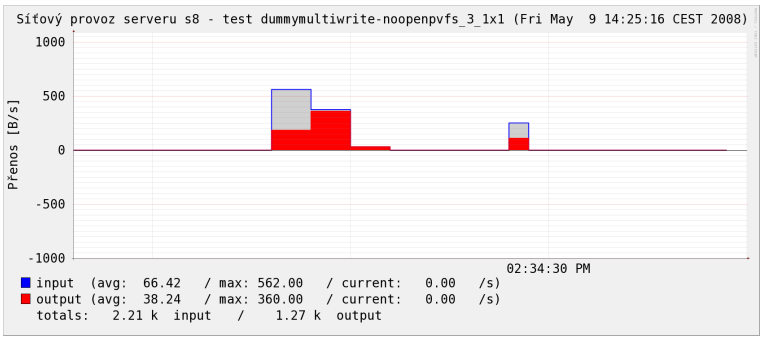


Obrázek 3.7: Síťové I/O metaserveru během testu opakovaného zápisu

jednotlivého MB se vždy přešlo znovu na začátek. Jak ukazuje graf 3.8, metaserver vypověděl síťovou činnost pouze v momentu otevření a zavření souboru, kdy odpověděl na požadavek o metainformace pro daný soubor při otevření a zapsal nový čas[9] poslední editace při zavření. Tento test trval stejně dlouho jako při zápisu GB souboru.

Test opakovaného zápisu

PVFS se při testu appendu osvědčil lépe než GlusterFS. Pokud do jednoho souboru zapisovalo více klientů ve stejnou chvíli 1 MB, nedošlo ke ztrátě dat a data byla uspořádána za sebou a neprolínala se po KB blocích jako v případě GlusterFS. Nicméně během testů připisujících každým klientem 1 GB, již ke ztrátě dat došlo. PVFS tedy také tuto součást POSIXu nedrží. Na druhou stranu, PVFS si ani neklade za cíl dodržovat přesně POSIX, jak přiznávají jeho autoři, a přitom se alespoň při práci s menším množstvím dat podle ní choval.



Obrázek 3.8: Síťové I/O metaserveru při opakovaném zápisu bez opětovného otevírání/zavírání souboru

Test výpadku

Odolnost proti výpadku jsem vyzkoušel stejným způsobem jako u GlusterFS. Výsledek byl taktéž stejný. Přesun dat se vždy pozastavil a po znovupřipojení dataserveru pokračoval. Na rozdíl od GlusterFS se však přenos zastavil, vypojil-li jsem libovolný z dataserverů, protože data přenášeného souboru se nacházela v celém clusteru.

[9] Samozřejmě mimo jiných případných informací...

Přenášelo-li se několik menších souborů, odpojení metaserveru způsobilo pozastavení až během dokončování kopírování aktuálně přenášeného souboru. Po opětovném připojení metaserveru vše opět pokračovalo.

Podobně jako u dataserverů, byly-li přítomny dva (více jsem nezkoušel) metaservery, přenos se pozastavil i s odpojením jednoho z nich, protože metainformace měly rozloženy oba dva rovnoměrně.

3.6 Gfarm Grid File System

Instalace

Gfarm je standardně k dispozici formou předkompilovaných RPM balíčků, které bylo možné využít pro instalaci na CentOS 5.1. V době testování byla již dostupná nová verze Gfarmu – verze 2.0.0. Vhledem k tomu, že se podle[3] jednalo o významnější skok oproti poslední předchozí verzi 1.4.1, chtěl jsem do testovacího procesu zapojit verzi 2.0.0. Pro tuto verzi byl však Gfarm dostupný pouze formou zdrojových kódů, instalaci jsem tedy provedl jejich přeložením a spuštěním.

Opět stačilo spustit trojici příkazů `./configure`, `make` a `make install`. Jako databázový backend metaserveru jsem pro testování zvolil PostgreSQL. Kromě toho nebo OpenLDAP je Gfarm závislý pouze na knihovně OpenSSL. Ta byla v našem systému již přítomná, protože jsem však prováděl instalaci ze zdrojových archívů, bylo třeba doinstalovat ještě *devel* variantu jejího balíčku.

Jelikož cílem bylo připojit Gfarm do jmenného prostoru klientských počítačů, bylo třeba naistalovat *GfarmFS-FUSE*. Ten je ve své verzi 2 k dispozici opět pouze v nepředkompilované formě. V adresáři se zdrojovým kódem *GfarmFS-FUSE* stačilo opět spustit `./configure`, `make` a `make install`, bez jakýchkoli parametrů.

Podobně jako v případě instalace PVFS byl adresář se zdrojovými kódy uložen na všemi počítači sdíleném NFS. Stačilo tedy provést kompilaci zdrojových archívů na jednom počítači a na ostatních pomocí skriptu pro vzdálené spuštění provést v patřičných adresářích `make install`.

Konfigurace

Gfarm jsem nejprve nastavoval pro konfiguraci obsahující jeden dataserver, pričemž metaserver běžel na stejném počítači.

Jak jsem již napsal v podkapitole 2.4.2, nejprve bylo třeba nakonfigurovat metaserver. Na patřičném počítači jsem spustil konfigurační program *config-gfarm*, který vygeneroval soubory `/usr/local/gfarm2.conf` a `/usr/local/gfmd.conf`. `Gfarm2.conf` slouží jako konfigurační soubor clusteru a je třeba ho kopírovat do stejného umístění na všech počítačích, které se ho jakýmkoli způsobem účastní.

Dále bylo třeba vytvořit uživatele „_gfarmfs“, který slouží k autentifikaci mezi dataserverem a metaserverem. Dodanou utilitou *gfkey* jsem v domovském adresáři uživatele *_gfarmfs* vytvořil soubor `.gfarm_shared_key` obsahující sdílený klíč sloužící k výše zmíněné autentizaci.

Poslední krok této konfigurace spočíval v nastavení dataserveru. Protože se dataserver nacházel na stejném počítači jako metaserver, byly soubor `gfarm2.conf` a uživatel *_gfarm* již přítomní. Dále bylo nutné spustit program *config-gfsd* a jako parametr mu dodat cestu ke složce, kam se na daném počítači měla umisťovat data.

Ke spuštění Gfarmu stačilo spustit démony *gfsd* a *gfmd* nacházející se po instalaci v `/etc/init.d`. Na počítačích určených k roli klienta jsem použil nainstalované *GfarmFS-FUSE* a spustil program `gfarm2fs /mountpoint`, čímž došlo k připojení této instance Gfarmu do systému souborů.

Při pokusu o konfiguraci a spuštění Gfarmu jako clusteru obsahujícího více dataserverů jsem se setkal s problémy. Nejprve docházelo ke komplikacím souvisejícím zejména s autentizací klientů k dalším přidávaným dataserverům, což se mi podařilo vyřešit a Gfarm spustit. Gfarm však poté nefungoval správně, nové soubory se nedařilo do exportovaného Gfarmu zapisovat.

Třebaže jsem nápravě takového stavu věnoval značné úsilí a čas, jenž jsem měl k dispozici, se mi nakonec Gfarm na konfiguracích s více dataservery rozjet nepodařilo. Provedl jsem tedy kompletní sadu testů, nicméně pouze na konfiguracích majících jeden dataserver.

Nemohu říct, že by chyba byla způsobena Gfarmem, i když se to v podstatě nabízí. Je možné, že k tomuto problému přispěl nějaký faktor související konkrétně s naším clusterem, nebo i nějaká chyba na mé straně, kterou jsem přes veškerou snahu odhalit ji, přehlédl.

3.6.1 Prováděné testy

Na Gfarmu jsem provedl stejné testy jako u GlusterFS a PVFS, nicméně pouze na sestavách s jedním dataserverem. I tak však bylo možné z výsledků testů, studiem vlastností Gfarmu a analýzou vzniklých grafů rozebrat některé vnitřní pochody Gfarmu a použít je k porovnání s chováním zbylých dvou paralelních DFS.

3.6.2 Analýza výsledků

V tabulce 3.3 jsou uvedeny doby trvání vykonávaných testů ze strany klientů v sekundách. V konfiguraci „1 x 2“ je uveden průměr těchto časů.

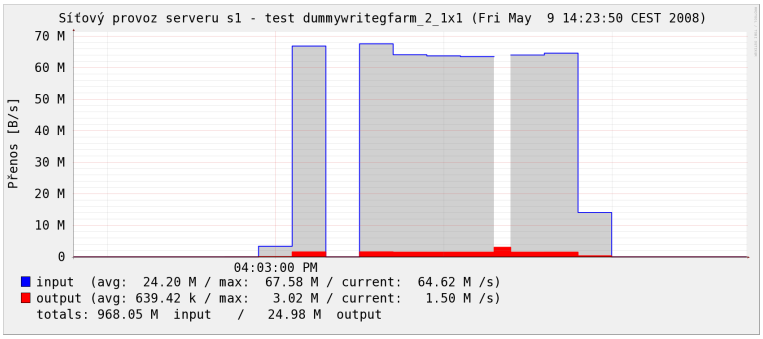
Test	1 x 1	1 x 2
Zápis	16,3	31
Čtení	12,2	29
Op. zápis	19,2	36
Op. čtení	< 1	< 1

Tabulka 3.3: Testy GlusterFS – uvedeno v sekundách

Zápis

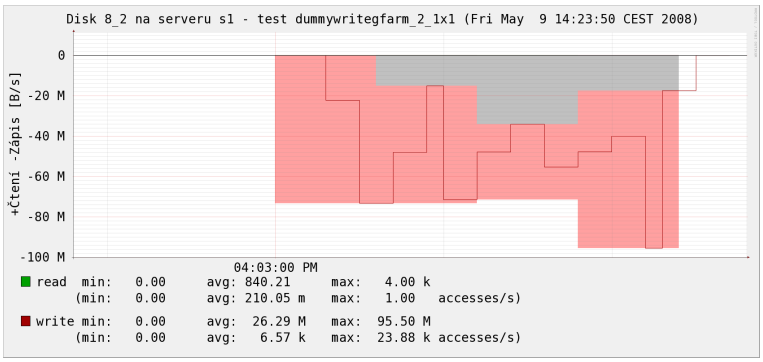
Při testu zápisu v sestavě „1 x 1“ dosáhl Gfarm v podstatě stejného výsledku jako GlusterFS. Graf 3.9 ukazuje, jak data přicházela dataserveru a byla uložena do cache o něco dříve, než disk dokončil zápis. Jak je vidět v grafu 3.10, ten pak ničím nebrzděn pokračoval v zápisu svou plnou rychlostí, dokud všechna data z cache nepřečetl.

Maximální propustnosti sítě však nebylo, podobně jako v případě GlusterFS, dosaženo. Délce trvání testu 16,3 s odpovídá průměrná rychlost vlastního přenosu okolo 63 MB/s. Omezující vliv tak měla nejspíš potřeba procesorového výkonu dataserveru. Ta byla do doby, než byl ukončen přenos sítí průměrně téměř padesátiprocentní a posléze, než disk dokončil zápis z cache, vyskočila na 100 %.



Obrázek 3.9: Činnost síťového rozhraní dataserveru během testu zápisu

Jestliže zapisovali v konfiguraci „1 x 2“ ve stejnou dobu dva klienti, test probíhal stejným způsobem a byl dvakrát tak pomalejší. Procesorové zatížení dataserveru u tohoto testu dosahovalo 100 % již po celou dobu testu (viz graf 3.11). Ve srovnání se zbylými dvěma DFS to tedy vypadá, že Gfarm má daleko větší procesorové nároky. I na klientské straně byla potřeba procesoru přibližně dvojnásobná - 10% - ve srovnání s potřebou GlusterFS a PVFS - 5% - u testu na stejné sestavě. Na vině mohl být metaserver, který běžel na stejném PC. To je však spíše nepravděpodobné vzhledem k malému objemu metadat, která se musela zpracovat.



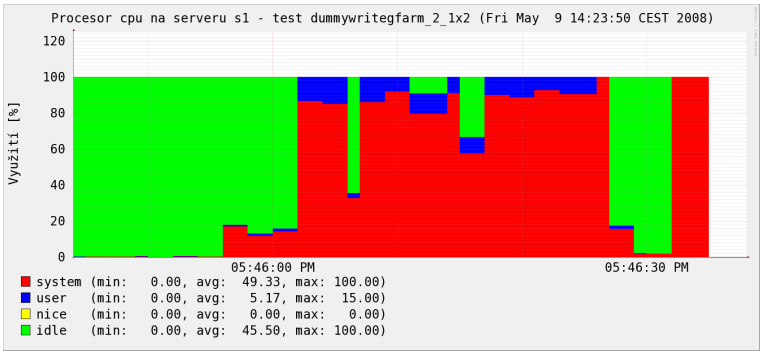
Obrázek 3.10: Činnost disku dataserveru během testu zápisu

Test opakovaného zápisu měl díky přítomnosti metaserveru u Gfarmu opodstatnění. Bylo dosaženo podobných výsledků jako u PVFS, doba potřebná k zápisu se ve srovnaní se standardním zápisem zvedla o 30% u konfigurace „1 x 1“ a 15% u konfigurace „1 x 2“. Ke zpoždění došlo i přesto, že metaserver běžel na stejném PC. Spíše než síťovou komunikací bylo v tomto případě zpoždění způsobeno potřebou metaserveru pracovat s pevným diskem.

I u tohoto DFS jsem provedl test zápisu 10 GB souboru k porovnání s testem přenosu 1 GB. Cílem byla opět alespoň částečná eliminace vlivu cache. Test jsem pro změnu provedl v konfiguraci „1 x 2“, svůj individuální 10 GB soubor tak zapisovali současně dva klienti. Test trval u obou 453 sekund – průměrná rychlost byla 22 MB/s, což je o 50% méně než průměrná rychlost přenosu 1 MB

Čtení

Čtení v konfiguraci „1 x 1“ probíhalo víceméně stejně rychle jako u PVFS i GlusterFS. Pokud se celý soubor četl z cache, byla omezujícím faktorem propustnost sítě. V konfiguraci „1 x 2“



Obrázek 3.11: Zátěž procesoru během testu zápisu dvěma klienty

bylo čtení asi o 4 sekundy „pomalejší“ než u stejného testu architektonicky srovnatelného PVFS.

Vzhledem k absenci testů na vyšších konfiguracích to nemohu tvrdit s jistotou, nicméně s přihlédnutím k vyšší procesorové náročnosti se domnívám, že Gfarm je o něco pomalejší než srovnatelné PVFS.

Při opakovaném čtení došlo k využití cache na straně klienta stejným způsobem, jako tomu bylo s klientem GluserFS. 1 MB soubor se načetl do paměti klienta a dalších 1023 opakování čtení proběhlo lokálně. Celý test trval méně než jednu sekundu, omezujícím faktorem zde tedy byla rychlost RAM klienta.

Zápis na konec souboru

Test zápisu na konec souboru byl opět proveden dvakrát, dvěma klienty se současně zapisoval 1 MB a v druhém případě 1 GB. Vzhledem k deklarované podpoře POSIXu dopadl první test dle očekávání. Data nebyla proložena a nacházela se za sebou. Během druhého testu byla data již zpřeházena, udržet konzistenci při appendech gigabytových rozměrů je zřejmě problematické, zvláště u distribuovaných FS. Ke ztrátě dat však nedošlo, výsledný soubor měl přesně 2 GB.

Test výpadku

Testování odolnosti proti výpadku bylo omezeno konfigurací s maximálně jedním dataserverem. Nebylo tak možno vyzkoušet skutečné chování v plně clusterovaném prostředí - zvláště v souvislosti s automatickou replikací souborů. Dále metaserver a jediný dataserver běžely oba na jenom počítači. Pokud jsem však odpojil síťový kabel serverového PC během kopírování v této konfiguraci, došlo k pozastavení a přenos po opětovném zapojení pokračoval.

Kapitola 4

Porovnání testovaných DFS

Obsahem této krátké kapitoly je závěrečný přehled vlastností zkoumaných paralelních DFS a také porovnání jejich užitečnosti pro nejrůznější aplikace použitelné především na půdě naší školy. Tento přehled se opírá o výsledky získané experimentální činností.

	GlusterFS	PVFS 2	Gfarm FS
Licence	GPL v .3	LGPL v. 2.1	BSD
Stabilní	Ano	Ano	Ano
Maximální velikost	Petabyty	Petabyty	Petabyty
Metaserver	Ne	Implementovaný společně s dataserverem	Démon *gfmd*, 1 v clusteru
Metaserver jako bottleneck	n/a	Ne, paralelně distribuovaný	Ne, metadata cache server
Chybová tolerance	Volitelná, translator *cluster/afr*	Ne	Ano, automatická
Striping	Volitelné, autory nedoporučováno	Ano	Ano
Připojení do VFS	FUSE	V linuxu, modul jádra	FUSE
Možnosti propojení	TCP/IP, infiniband	TCP/IP, infiniband, Myricom GM	TCP/IP
Výpadek dataserveru	pozastaveno / bez vlivu	pozastaveno vždy	pozastaveno / n/a
POSIX append, 1 MB	Ano, nesouvisle	Ano, souvisle	Ano, souvisle
POSIX append, 1 GB	Ne	Ne	Ano, nesouvisle

Tabulka 4.1: Srovnání některých testovaných DFS

Především všechny tři paralelní DFS, na jejichž testování jsem se zaměřil, patří mezi stabilní a do praktického nasazení použitelné distribuované souborové systémy. Má-li být jeden vybrán, záleží na potřebách konkrétního uživatele a na způsobu, jakým jeho klientské aplikace budou k DFS přistupovat.

Konstrukčně jsou testované DFS v několika ohledech rozdílné, zvláště pak GlusterFS.

K problematice rozdělení dat a zátěže na clusteru přistupuje svým specifickým způsobem. Místo stripování ukládá soubory na své servery vcelku a o rozdělení zátěže se stará scheduler. Těchto plánovačů je na výběr několik druhů dovolujících upravit metodiku rozložení souborů dle konkrétních potřeb. Díky modularitě umožněné použitím konceptu *translators* a schopnosti nejrůzněji je kombinovat v konfiguračním souboru se jedná o systém, jenž je ze všech tří testovaných nejlépe přizpůsobitelný.

GlusterFS bych nasadil především tam, kde by bylo třeba větším množství klientů zároveň přistupovat k velkému množství souborů. V rámci naší školy by vhodnou aplikací bylo ukládání multimediálních dat, například záznamů přednášek rozložených na menší množství serverů. GlusterFS by nejen zajistil jednotný jmenný prostor, ale i vhodně rozložil multimediální data. Vzhledem k přítomnosti kompletních souborů na různých serverech by bylo dosaženo vysokého výkonu při současném čtení nebo zápisu několika klientskými aplikacemi zároveň.

PVFS a Gfarm jsou si architektonicky podobnější. Oba rozkládají soubory na menší části, které potom ukládají na celém clusteru. S tím je spojena potřeba metaserveru, který obstarává informace potřebné k logické kompletaci souborů. Zvláště v případě výše uvedeného užití dosahují tyto dva DFS obecně nižšího výkonu, což je způsobeno nutnou síťovou, diskovou i procesorovou režií potřebnou ke kompletaci souborů.

Na druhou stranu, PVFS i Gfarm jsou vhodnější v situacích, kdy je potřeba pracovat s jedním velkým souborem, zvláště přistupuje-li k němu více klientů v jednu dobu - což je situace, představitelná např. během distribuovaných výpočtů. V případě GlusterFS by pak jeden server byl přetížený, zatímco ostatní by zahálely. PVFS i Gfarm mají soubory rozloženy rovnoměrně v celém clusteru, ani tedy příliš nezáleží na tom, zda-li je potřeba I/O práce soustředěna na jeden nebo více souborů. Platí za to však zmíněnou režií, která u GlusterFS odpadá. Pokud by bylo třeba pracovat s množstvím malých souborů, bude tato režie značně snížena. Budou-li mít používané soubory velikost menší jednoho bloku, budou vlastně rozloženy stejně jako v GlusterFS.

Mezi Gfarmem a PVFS – V rámci prováděných testů měl Gfarm výrazně větší procesorovou náročnost, z čehož zřejmě vyplývá i jeho předpokládaný částečně nižší výkon proti PVFS. Gfarm je také systémem plně POSIXovým, a proto má menší svobodu volby v implementačních otázkách. Na straně druhé však Gfarm pak jako jediný neztratil data při poněkud extrémním testu zápisu na konec souboru. Oproti PVFS je chybově tolerantní, automaticky vytváří repliky souborů, čímž je zvýšena odolnost proti ztrátě dat. PVFS repliky souborů neudržuje, snaží se ale maximálně snížit komplikace vzniklé výpadkem svou bezstavovostí vyplývající z mírného odklonu od přísného POSIXu. Problémem PVFS pak může být ztráta dat v celém systému při poruše jediného disku. Toto riziko se však dá značně snížit využitím RAID polí na datových serverech.

Práce mého kolegy[16] obsahuje podobný přehled jako je přehled uvedený na předchozí straně. Je tedy možné oba použít pro porovnání námi zpracovávaných systémů. Kolegův přehled navíc porovnává i rychlosti, které na konfiguraci „4 x 4“ dosahovaly jemu přiřazené DFS. Vzhledem k odlišným vlastnostem GlusterFS[1] a problémům s prováděním kompletní sady testů Gfarmu nebylo možné toto porovnání vložit i do mého přehledu. Bude-li třeba porovnat výkonnostní aspekty mnou přidělených systémů s Lustre, tzn. „nejlepším“ systémem kolegy, je možné využít tabulek a informací z předchozí kapitoly.

[1] Především absence stripingu měla zásadní vliv na chování GlusterFS v testech s více klienty.

Kapitola 5

Závěr

Cílem této práce bylo porovnání vlastností vybraných paralelních DFS a jejich otestování na půdě školy. V rámci semestrálního projektu jsem nejprve z veřejně dostupných zdrojů nastudoval informace o uvažovaných systémech a připravil sadu testů jejich vlastností. Kritéria, podle kterých jsem distribuované FS porovnával, byla ovlivněna především možnostmi, jak by mohly být tyto systémy využity v praxi naší fakulty.

Experimentální činnost probíhala ve školní laboratoři, hardwarová výbava byla zapůjčena fakultou. Na přípravě a konfiguraci testovacího clusteru a při analýze výsledků jsem spolupracoval s kolegou Martinem Tomcem, který paralelně se mnou zpracovával další DFS ve své bakalářské práci[16]. Vzhledem k tomu, že jsem testoval více souborových systémů, nastavoval jsem je standardním, pro každý systém nejčastěji používaným způsobem. U GlusterFS a PVFS 2 jsem provedl kompletní sadu testů na nejrůznějších sestavách clusteru. Při práci s Gfarm FS jsem narazil na komplikace a kompletní sadu testů jsem provedl jen na některých uvažovaných sestavách.

Během analýzy výsledků testů a získaných grafů jsem se zaměřil především na určení hlavních faktorů omezujících výkony jednotlivých systémů. Na závěr jsem porovnal testované souborové systémy. Zohlednil jsem informace nabyté během experimentální činnosti s nimi. Jak se ukázalo, všechny testované systémy byly dostatečně stabilní a použitelné v rámci některé z aplikací ve škole.

GlusterFS se díky své architektuře, nepoužívající striping, ukázal být vhodný pro ukládání množství větších souborů s nahodilým přístupem klientů, což se děje například v systému pro stahování záznamů přednášek. PVFS a Gfarm by byly vhodné v systémech s častým přístupem klientů k méně souborům. PVFS přináší oproti Gfarmu vyšší I/O výkon, Gfarm chybovou toleranci a lepší podporu pro aplikace vyžadující splnění normy POSIX.

V budoucnu by dále bylo možné vybrat jeden z testovaných DFS již se zaměřením na konkrétní aplikaci. V takovém případě by bylo vhodné systém detailně otestovat nejen v dané sestavě, ale i prozkoumat vliv jeho různých konfigurací na vybranou aplikaci.

Literatura

[1] Ananth Devulapalli. File creation strategies in a distributed metadata file system. In *IPDPS '07*, 2007.

[2] Inc Myricom. *The GM Message Passing System*, 2000.

[3] Youhei Morita Osamu Tatebe, Satoshi Sekiguchi. Gfarm v2: A grid file system that supports high-performance distributed and parallel data computing. Technical report, Department of Computer Science, University of Tsukuba, 2007.

[4] FTP server. Zdroj stažení RPM balíčků GlusterFS v. 1.3.7. `http://www.gluster.org/docs/index.php/GlusterFS`.

[5] WWW stránky. GlusterFS Translators v1.3. `http://www.gluster.org/docs/index.php/GlusterFS_Translators_v1.3`.

[6] WWW stránky. Message Passing Interface. `http://www-unix.mcs.anl.gov/mpi/`.

[7] WWW stránky. Online dokumentace GlusterFS. `http://www.gluster.org/docs/index.php/GlusterFS`.

[8] WWW stránky. Přehled publikací věnovaných dCache. `http://www.dcache.org/manuals/index.shtml`.

[9] WWW stránky. Remote Direct Memory Access. `http://www.networkworld.com/details/5221.html`.

[10] WWW stránky. Stránka projektu Gluster. `http://www.gluster.org/`.

[11] WWW stránky. Webové stránky Gfarm FS. `http://datafarm.apgrid.org/`.

[12] WWW stránky. Webové stránky MogileFS. `http://www.danga.com/mogilefs/`.

[13] WWW stránky. Zdroj obrázku obecného paralelního DFS. `http://edu.arabsgate.com/showthread.php?t=494149`.

[14] PVFS Development team. *PVFS Developer's Guide*, 2003.

[15] PVFS Development team. *A Quick Start Guide to PVFS2*, 2008.

[16] Martin Tomec. *Porovnávání vlastností distribuovaných souborových systémů.* FIT VUT v Brně, 2008. Bakalářská práce.

[17] WWW stránky. Online dokumentace gfarm fs. `http://datafarm.apgrid.org/document/`.

[18] Satoshi Sekiguchi Yusuke Tanimura, Yoshio Tanaka. Performance evaluation of gfarm version 1.4 as a cluster filesystem. Technical report, Department of Computer Science, University of Tsukuba, 2007.