FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Rozšíření funkce simulátoru disků DiskSim
Student:	Jana Berušková
Vedoucí:	Ing. Jiří Kašpar
Studijní program:	Informatika
Studijní obor:	Informační technologie
Katedra:	Katedra počítačových systémů
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

Analyzujte kód simulátoru DiskSim, zaměřte se na implementaci diskových sad RAID1 a 5. Rozšiřte simulátor o zabezpečení pomocí Reed-Solomonova schématu a možnosti simulace poruchy disku a rekonstrukce diskové sady po náhradě porouchaného disku novým.

Funkčnost doplňku podrobte testům použitelnosti podle pokynů vedoucího práce.

Seznam odborné literatury

J. S. Plank: A Tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-like Systems. Technical Report CS-96-332, University of Tennessee, 1996.
Robinson, Stewart: Simulation: The Practice of Model Development and Use. John Wiley & Sons, 2004
Harry Perros: Computer Simulation Techniques: The definitive introduction! Online: http://www4.ncsu.edu/ hp/files/simulation.pdf 2009

prof. Ing. Róbert Lórencz, CSc.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 15. prosince 2017

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Rozšíření funkce simulátoru disků DiskSim

Jana Berušková

Katedra počítačových systémů
Vedoucí práce: Ing. Jiří Kašpar

10. ledna 2019

Poděkování

Chtěla bych moc poděkovat svému vedoucímu práce Ing. Jiřímu Kašparovi za jeho trpělivost a pomoc při vytváření této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 10. ledna 2019 .....

Odkaz na tuto práci

Berušková, Jana. *Rozšíření funkce simulátoru disků DiskSim.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

Abstrakt

Tato bakalářská práce analyzuje, jak diskový simulátor DiskSim simuluje disková pole typu RAID a rozšiřuje ho o funkce, které mu umožní simulovat práci s daty i při výpadku jednoho a více disků. Problém zabezpečení dat je vyřešen RAID5 i RAID6 pomocí Reed-Solomonova schématu, který umožňuje rekonstrukci dat na více discích.

Hotový simulátor se pak může používat při výuce o datových úložištích i v praxi pro porovnávání různých konfigurací.

Klíčová slova DiskSim, rozšíření diskového simulátoru DiskSim, datová úložiště, diskové pole, RAID5, RAID6, Reed-Solomonovo schéma

Abstract

This bachelor thesis analyzes how disk simulator DiskSim simulates RAID arrays and extends it with functions that allow it to simulate the work with the data even when one or more disks are broken. The data security issue

is resolved in RAID5, and RAID6 by using the Reed-Solomon scheme that allows the data to be reconstructed on multiple disks.

The completed simulator can then be used in the education about data storage and in practice to compare different configurations.

Keywords DiskSim, extension of disk system simulator DiskSim, data storage, disk array, RAID5, RAID6, Read-Solomon scheme

Obsah

Seznam obrázků

Úvod

Diskové simulátory se v praxi mohou používat pro několik účelů. Běžný uživatel si v nich může nasimulovat předpokládanou zátěž svého budoucího počítačového systému a podle výsledků simulace si pak může vybrat a pořídit jednotlivé komponenty, které budou nejlépe vyhovovat potřebám jeho aplikace. Jejich další využití může být také pro studijní účely. Simulátor může být používán na technických školách, kde se pomocí něj studenti mohou učit principy ukládání dat v počítačových systémech a přístupu k nim.

Cílem mé práce je analýza práce diskového simulátoru DiskSim a rozšíření jeho implementace o funkce, které umožní simulovat i práci v degradovaném režimu a práci s daty při výpadku jednoho nebo vice disků pomocí Reed-Solomonova schématu.

V první části práce stručně představuji simulátor DiskSim a jakým způsobem fungují disková pole typu RAID. Dále pak rozebírám princip Reed-Solomonova schématu a jeho použití pro zabezpečení dat proti chybám.

V prostřední části se pak detailněji zaměřuji na DiskSim. Vysvětluji, jak může uživatel simulátor nakonfigurovat a jaké z něj dostane výsledky. Dále také popisuji, jak simulátor sám pracuje. Jakým způsobem simulace probíhají a jaké jsou k tomu potřeba datové struktury.

V praktické části pak popisuji samotný návrh rozšíření a jeho implementaci. Rozšířením jsou funkce, které umožní simulovat i práci v degradovaném režimu v konfiguraci RAID5 a RAID6. Tyto nové funkce mají data zabezpečená pomocí obecného Reed-Solomonova schématu.

Tato práce navazuje na semestrální projekt studenta Kamila Jakuboviče z předmětu Datová úložiště [1], který zpřehlednil simulátor pro uživatele a vytvořil šablony a skripty pro snadnější nastavení konfigurace jednotlivých

simulací. V kombinaci s mým rozšířením se pak DiskSim stává komplexním, ale zároveň uživatelsky přívětivým simulátorem i pro domácí použití.

Kapitola 1

Úvod do problému

Na začátku této práce je na místě vysvětlit pár základních pojmů, které jsou potřeba pro její pochopení.

1.1 Co je simulace

Obecně můžeme simulaci definovat jako napodobeninu systému [2]. V tomto kontextu existují dva druhy simulace tzv. *statické* a *dynamické*. Statické simulace imitují nějaký systém pouze v jednom časovém bodě a dynamické zase simulují systém v průběhu nějakého časového období [3].

Simulace mají mnoho využití. Běžně se s nimi setkáváme například v domácnostech. Vidíme je každý den v televizi, když se díváme třeba na předpověď počasí. Nebo se i my sami můžeme stát jejich součástí pomocí herních konzolí, které napodobují nejrůznější aktivity [2].

Dále jsou velice hojně využívané například společnostmi při plánování nejen struktury jejich nových podniků, ale i ke zlepšování těch současných [2]. Pro takové společnosti je výhodné vyvinout simulaci, která pomůže odhalit případné budoucí nedostatky navržených systémů, aby se mohly včas odstranit.

Simulace se využívají právě proto, že se správně zadanými parametry jsou schopné předpovídat budoucí chování daného systému. Díky tomu jsme pak schopní, pomocí úpravy daných parametrů, své systémy upravit tak, aby co nejlépe odpovídaly požadovanému výsledku.

1.2 Co je DiskSim

DiskSim je efektivní, přesný a snadno konfigurovatelný diskový simulátor. Je napsán v jazyce C nad POSIX rozhraním. V samostatných modulech simuluje chování všech komponent ve zpracování vstupně výstupních operací včetně ovladačů, sběrnic, adaptérů, řadičů a samozřejmě i disků [4].

Uživatel má možnost nakonfigurovat si vlastní topologie všech výše uvedených komponent. Navíc může zařadit jednotlivé disky do různých diskových polí. DiskSim podporuje i většinu polí typu RAID.

Svoji nadefinovanou topologii pak uživatel může testovat při různých zátěžích vstupně výstupních operací. Tyto zátěže může uživatel externě zaznamenat a pak je v simulátoru spustit. DiskSim podporuje několik formátů pro zaznamenanou zátěž a některé se do něj dají ještě přidat [4].

Další možností jak vytvořit požadovanou zátěž v simulátoru je použití vestavěného generátoru zátěže. Ten si může uživatel nastavit přímo v konfiguračním souboru, aby vytvářel požadavky na čtení nebo zápis na disk podle jeho preferencí.

DiskSim pak během simulace zaznamenává mnoho různých údajů o výkonu všech komponent v dané topologii. Z těchto údajů pak vytváří statistiky, které jsou hlavním výstupem celé simulace. Uživatel má možnost v konfiguračním souboru nadefinovat, které statistiky ho zajímají a které mu má DiskSim poskytnout [4].

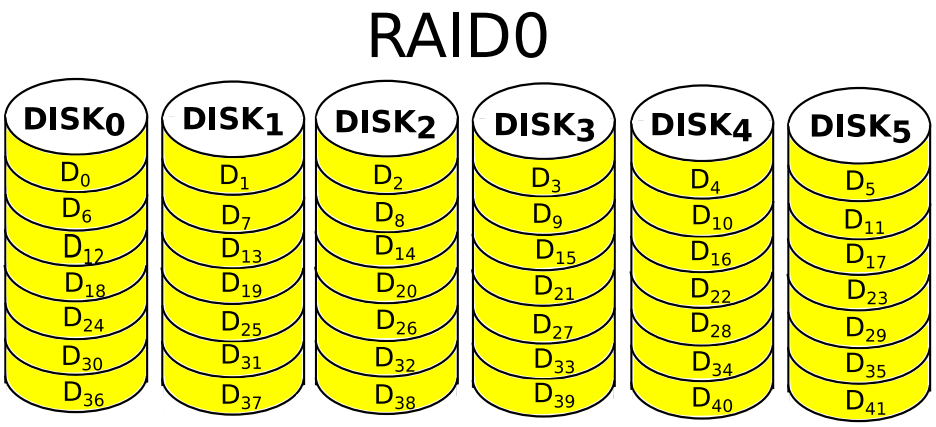
1.2.1 Historie simulátoru DiskSim

DiskSim byl vyvíjen na University of Michigan a následně rozšířen na Carnegie Mellon University od roku 1992. V roce 2008 byla vydána zatím poslední verze - DiskSim-4.0 [5]. Tím však jeho vývoj neustal, neboť je stále rozšiřován pomocí přidaných patchů. Například Microsoft vytvořil model pro SSD disky a v roce 2015 zveřejnil Western Digital upravenou verzi 4.0, která podporuje 64-bitovou adresaci a 64-bitová čísla logických bloků (pro disky vetší než 2TB) [6].

Tento simulátor byl také využit v celé řadě publikovaných studií z oblasti úložných systémů pro modelování výkonu různých konfigurací a architektur datových úložišť i nových technologií pro ukládání dat [5].

V současné době se simulátor DiskSim používá například na fakultě informačních technologií ČVUT, kde slouží studentům pro výuku datových úložišť.

Zároveň je veřejně dostupný, takže ho může použít každý, kdo by chtěl porovnávat různé konfigurace úložných zařízení, a podle výsledných dat si vybrat, jaká zařízení by si měl pořídit, aby nejlépe sloužila jeho potřebám.



Obrázek 1.1: Rozdělení dat do chunků v RAID0

1.3 Disková pole typu RAID

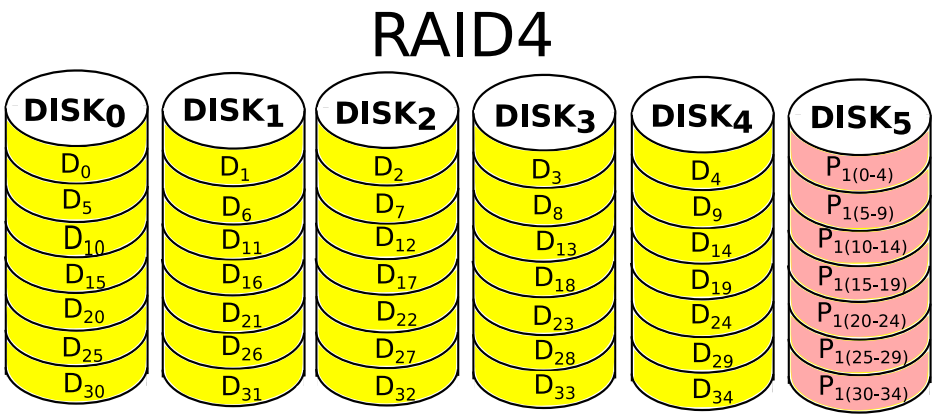
Disková pole jsou dnes alternativou k datovým úložištím založeným na jednom velikém disku. Takové úložiště má sice velkou kapacitu, ale nemůže nabídnout takový výkon jako diskové pole [7]. V diskovém poli se velký objem dat může rozložit na několik disků a tím se dá zrychlit doba zápisu i čtení.

Pokud však data ukládáme na více disků, pravděpodobnost že o ně přijdeme se tím zvyšuje. Pokud máme data pouze na jednom disku, existuje určitá pravděpodobnost, že se tento disk porouchá. Pokud máme disků více, může se porouchat kterýkoli z nich a to zvyšuje celkovou pravděpodobnost ztráty dat [8].

Proto se začala používat disková pole typu RAID (Redundant Array of Inexpensive/Independent Disks). Ty přidávají do pole disky navíc, na kterých se ukládají redundantní data, díky kterým můžeme ztracená data z případného poškozeného disku obnovit [8]. Různé typy RAIDů mají různou strategii pro obnovu dat.

Základním typem je RAID0. Ten žádné disky navíc nepřidává a neexistuje žádná možnost obnovy ztracených dat. Používá se ke zvýšení propustnosti při práci s daty. Data jsou totiž rozdělena mezi všechny disky pole a ukládána do tzv. `chunků`. Chunk je malá část disku o určité velikosti. Tato velikost určuje, kolik souvislých bloků dat je na každém disku. To umožňuje při zápisu nebo čtení velkého objemu dat využít i paralelní přístup. Toto rozdělení je ukázáno na obrázku 1.1.

Dalším typem je RAID1, který ukládá přesné kopie dat na redundantní disky. Pro každý datový disk, musí být přidán nejméně jeden redundantní.



Obrázek 1.2: Paritní disk v RAID4

Toto řešení, tedy vyžaduje velký počet fyzických disků v poli.

RAID4 obsahuje vždy jeden redundantní disk. Na tomto disku se ukládá tzv. parita. Parita se vypočítává z dat uložených v tzv. `redundantní sadě`. Redundantní sada je skupina chunků, přičemž se z každého disku počítá právě jeden chunk. Obrázek 1.2 ukazuje schéma diskového pole s paritou pro každou redundantní sadu na jednom disku.

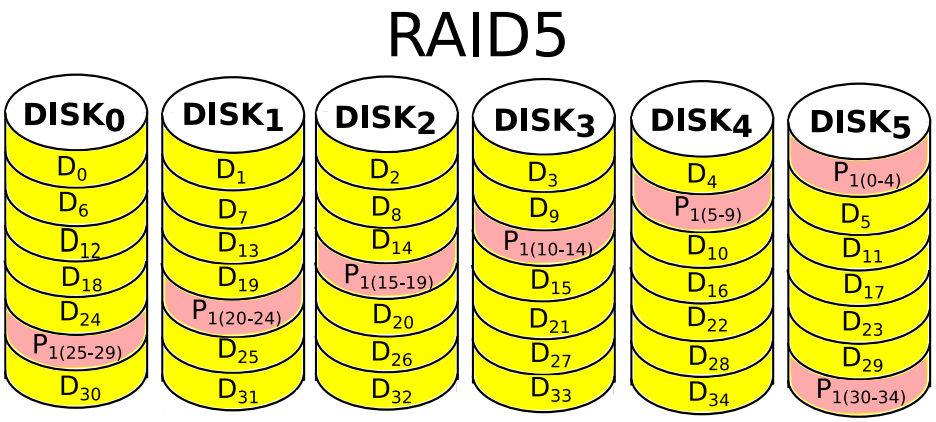
Pokud by jeden disk přestal fungovat, pomocí ostatních datových disků a parity se dají původní data dopočítat. Parita se však musí přepočítat po každém zápisu na některý z disků.

RAID5 obsahuje stejně jako RAID4 právě jeden redundantní disk a data jsou v každé redundantní sadě také zabezpečena pomocí parity. Aby však nedocházelo k přetěžování jednoho disku, je parita v RAID5 distribuovaná mezi všechny disky [7], jak je ukázáno na obrázku 1.3.

RAID6 se chová stejně jako RAID5 s tím rozdílem, že má dva redundantní disky. Data v paritních chuncích jedné sady se počítají nezávisle na sobě, každý pomocí jiné funkce. Díky tomu jsme schopni obnovit data až ze dvou porouchaných disků.

1.4 Reed-Solomonovo schéma a zabezpečování dat

Reed-Solomonovy kódy jsou nebinární cyklické samoopravné kódy [9]. Takové kódování se obecně používá ke zvýšení spolehlivosti komunikace pomocí redundantních dat, díky nimž dokážeme detekovat a opravit chybná data do původní podoby [10].



Obrázek 1.3: Distribuaovaná parita v RAID5

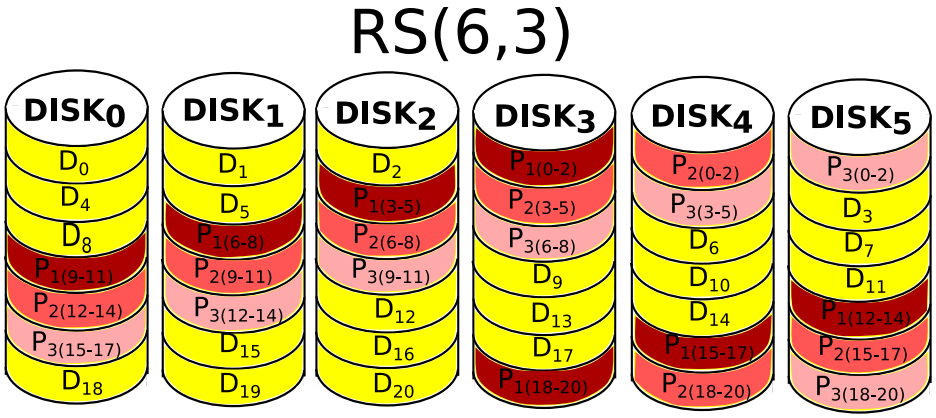
Obecné Reed-Solomonovo schéma se může použít i při zabezpečování dat v diskových polích. `RS(n,k)` má k dispozici `n` fyzických disků a na `k` discích jsou uložena uživatelova data. Pak nám zbývá `n-k` disků pro vypočítaná zabezpečovací data. Na každém z těchto disků jsou uložena data, která se pomocí různých funkcí vypočítají ze všech datových disků [10].

Data na jednotlivých discích jsou rozděleny na chunky. Zabezpečovací funkce se pak aplikují na chunky stejné úrovně na všech datových discích a vypočítají tak zabezpečovací data těchto chunků datových disků. Důležité je, že pro každý zabezpečovací disk, musí existovat jiná funkce vypočítávání dat. Díky tomu jsme pak schopní obnovit data až z `n-d` porouchaných disků [8].

Podobně jako v RAID5 a RAID6 jsou zabezpečovací data rozdistribuována mezi všechny fyzické disky v poli. Obrázek 1.4 ukazuje diskové pole se zabezpečením pomocí Reed-Solomonova schématu.

Jelikož Reed-Solomonovo schéma platí obecně pro jakýkoli počet redundantních disků, můžeme ho použít místo některých RAIDů. Pro RAID0 můžeme použít `RS(n,n)`, pro RAID5 pak `RS(n,n-1)` a pro RAID6 by schéma bylo `RS(n,n-2)`.

Bude nám tedy stačit jen jedna obecná funkce pro všechny tyto RAIDy a pomocí parametrů pro počet disků je budeme rozlišovat.



Obrázek 1.4: Zabezpečení dat pomocí Reed-Solomonova schématu

Kapitola 2

Rozbor problému

Po rozebrání těchto důležitých pojmů mohu nyní přistoupit k detailnějšímu popisu DiskSimu a jeho konfigurace.

2.1 DiskSim

Jak zde už bylo řečeno, DiskSim se skládá z několika různých komponent, které si uživatel může libovolně poskládat do požadované topologie. Tuto topologii musí nadefinovat v konfiguračním souboru pro konkrétní simulaci. Dále si zde může nadefinovat, jaké statistiky by měl DiskSim vytisknout ve výstupním souboru.

2.1.1 Komponenty simulátoru DiskSim a jejich topologie

Simulátor DiskSim obsahuje čtyři hlavní komponenty. Těmi jsou ovladače zařízení, sběrnice, řadiče a samotná disková zařízení [4].

Na vrcholu každé topologie musí být právě jeden ovladač. V konfiguračním souboru jsou jeho parametry nastavovány pomocí struktury `disksim_iodriver`. Na něj je vždy navázán řadič. Ten je v konfiguraci nastavován pomocí `disksim_ctrl`.

Za řadičem pak může být ještě jeden řadič a nebo už přímo diskové zařízení. Jeho parametry se ukládají do struktury `disksim_disk`. Konfiguračních parametrů pro jednotlivé typy disků je mnoho, a proto se většinou nachází ve zvláštním souboru a do konfiguračního souboru se zařazují pomocí direktivy `source` [4].

Mezi všemi předchozími komponentami se pak vždy musí nacházet sběrnice, která se nastavuje ve struktuře `disksim_bus`. Ta propojuje vždy jednu komponentu nadřazenou s jednou nebo více komponentami podřazenými.

Na jednu sběrnici může navazovat až patnáct podřazených zařízení. Naopak na jeden řadič pak můžou navazovat až čtyři sběrnice (ovšem varianta s více sběrnicemi za řadičem dosud nebyla řádně otestována) [4].

V uživatelské příručce k simulátoru DiskSim je uveden příklad zápisu topologie a k ní odpovídající obrázek zapojení komponent [4]. V manuálu však byl obrázek chybný a já zde na obrázku 2.1 uvádím správnou podobu topologie.

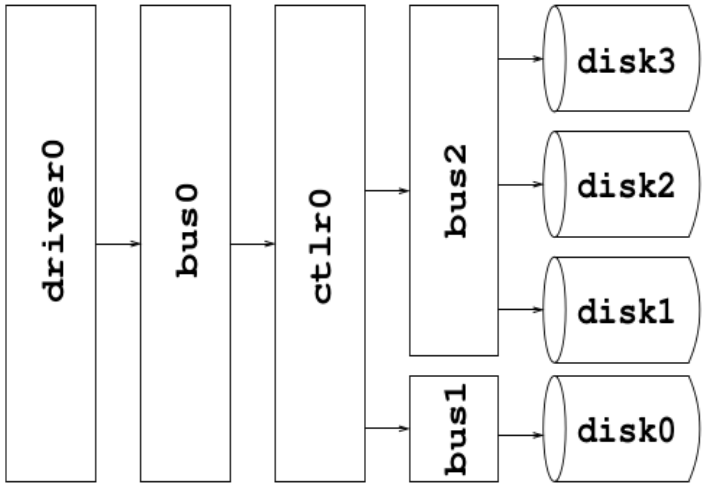
```
topology disksim_iodriver driver0 [
   disksim_bus bus0 [
      disksim_ctlr ctlr0 [
         disksim_bus bus1 [
            disksim_disk disk0 []
         ] # end of bus1
         disksim_bus bus2 [
            disksim_disk disk1 []
            disksim_disk disk2 []
            disksim_disk disk3 []
         ] # end of bus2
      ] # end of ctlr0
   ] # end of bus0
] # end of system topology
```

2.1.2 Vstupní a výstupní parametry simulace

DiskSim potřebuje ke spuštění simulace dva druhy vstupních dat. Jedním z nich jsou vstupně výstupní operace, které by měly v simulaci proběhnout. Ty se mohou načíst buď z externího souboru nebo přímo ze standardního vstupu anebo se mohou nechat vygenerovat simulátorem pomocí vestavěného generátoru zátěže.

Druhým nepostradatelným vstupním souborem je pak konfigurační soubor. Ten se skládá ze 3 různých typů dat: bloků pro nastavení parametrů simulace, instanciace všech fyzických zařízení a popisu topologie systému [4].

V blocích uzavřených v závorkách {} jsou nastavené nejrůznější parametry potřebné pro danou simulaci. Povinnými bloky pro každý konfigurační



Obrázek 2.1: Opravený příklad topologie komponent v DiskSimu

soubor jsou bloky `Global` a `Stats`. V bloku `Stats` se nastavuje informace, které statistiky mají být během simulace shromažďovány [4].

Jediným povinným prvkem bloku `Global` je parametr `Stat definition file`, který udává jméno dalšího vstupního souboru, potřebného ke spuštění simulace. V něm se popisují statistiky, pro které nestačí pouze jedna hodnota, ale výčet několika hodnot. V tomto souboru se nastavují rozsahy hodnot daných statistik, které se budou zaznamenávat ve výstupním souboru.

Pomocí dalších bloků jsou nastavovány parametry i jednotlivým použitým komponentám. Z těchto komponent se pak sestaví topologie celého systému, jak jsem již popisovala výše.

Před použitím v topologii však musí být každá použitá komponenta instanciována a pojmenována. K tomu slouží sekce konfiguračního souboru ve tvaru `instantiate [ <jméno_zařízení>, <jméno_zařízení>, ... ] as <název_komponenty>`, kde název komponenty je název daný komponentě při specifikaci jejích parametrů [4]. V seznamu zařízení může být použit i rozsah, pokud se zařízení jmenují stejně a liší se pouze číselnou příponou.

Další částí v konfiguračního souboru jsou bloky popisující struktury `disksim_logorg`. V těchto strukturách se definují disková pole. Každé čtení nebo zápis musí vždy padnout do nějakého pole, takže alespoň jedna tato struktura musí v konfiguračním souboru být [4]. Pokud nechceme, aby byl disk součástí pole, můžeme ho zařadit do této struktury jako jediný disk.

Pokud by v simulaci měly být použity vstupně výstupní operace gene-

rované vestavěným generátorem zátěže, nacházejí se v poslední části konfiguračního souboru ještě bloky, nastavující právě tento generátor. Tyto bloky se nazývají `Proc` a `Synthio` [4].

Kompletní seznam parametrů, které se dají v konfiguračním souboru nastavit, se nachází v uživatelské příručce DiskSimu. Konkrétní příklad jednoho celého konfiguračního souboru uvádím v příloze A této práce.

Hlavním výstupem simulace je textový soubor, ve kterém jsou zapsány všechny statistické údaje o výkonu jednotlivých komponent během simulace, které se podle konfigurace měly zaznamenat. Pokud byla zátěž generována během simulace, jsou zde zapsané i statistiky o proběhlých vstupně výstupních operacích.

Na začátku výstupního souboru je okopírovaná konfigurace dané simulace a na ní pak navazují samotné statistiky. Každá statistika má své unikátní jméno. Ve většině statistik stačí uvést jednu hodnotu ve tvaru `<jméno>: <hodnota>`. V některých případech je jedna hodnota nedostatečná k popisu dané statistiky, takže se namísto toho uvádí čtyři souhrnné statistiky: průměrná hodnota, směrodatná odchylka, maximální hodnota a výčet hodnot v rozsahu nastaveném ve vstupním souboru pro definice těchto statistik [4].

Uživatel má ještě možnost zaznamenat informace o každé jednotlivé vstupně výstupní operaci, tedy o každém požadavku na čtení nebo zápis. Tyto informace se ukládají do souborů jejichž jména může uživatel zadat do parametrů `Output file for trace of I/O requests simulated` a `Detailed execution trace` v `Global` bloku v konfiguračním souboru.

2.1.3 Spuštění simulace

Simulátor DiskSim se spouští pomocí příkazové řádky pro každou požadovanou simulaci zvlášť. Kromě zavolání samotného DiskSimu vyžaduje příkaz ještě pět dalších povinných argumentů a dále libovolný počet volitelných argumentů, kterými se dají přepsat některé parametry uložené v konfiguračním souboru [4].

Spuštění jedné simulace v CLI pak tedy vypadá takto:

```
disksim <parfile> <outfile> <tracetype> <tracefile> \
        <synthgen> [ par_override ... ]
```

- `disksim` - název spustitelného programu DiskSim
- `parfile` - název konfiguračního souboru
- `outfile` - název výstupního souboru

- `tracetype` - formát, v jakém jsou uloženy vstupně výstupní operace, které se mají v simulaci provést
- `tracefile` - název souboru se zaznamenanými vstupně výstupními operacemi (pokud je zadáno `stdin`, informace se berou ze standardního vstupu; pokud je zadána `0`, měl by být použit vestavěný generátor)
- `synthgen` - udává zda, se v simulaci má použít vestavěný generátor vstupně výstupních operací (pokud je zadána `0`, měl by být vyplněný `tracefile`)
- `par_override` - tyto argumenty přepisují hodnoty daných parametrů v konfiguračním souboru - jeden parametr se přepisuje vždy trojicí argumentů `<komponenta> <název_parametru> <nová_hodnota>`

2.1.4 Chybějící funkce

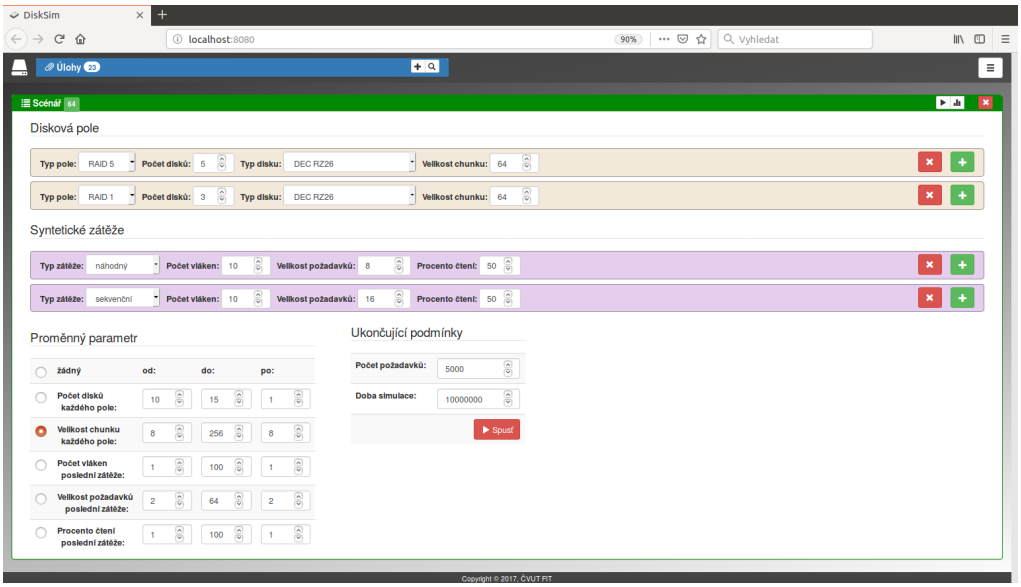
DiskSim zařazuje všechna disková zařízení do diskových polí. Jsou zde naimplementována mimo jiné i pole typy RAID0, RAID1, RAID4 a RAID5. Zatím však pracuje pouze s funkčními disky a neřeší případy, kdy nějaký disk z pole vypadne.

Je tedy třeba doimplementovat do něj funkce, které budou obstarávat simulace rekonstrukce dat na poškozeném zařízení. V praxi se také může stát, že se porouchá i více než jeden disk najednou. Proto by nové funkce měly být schopné pracovat s poli, které dokáží rekonstruovat data na více zařízeních. K tomuto účelu dobře poslouží zabezpečení dat pomocí Reed-Solomonova schématu.

S tím souvisí i úprava konfiguračních parametrů. V obecném Reed-Solomonově schématu, si uživatel může sám vybrat, kolik disků v jeho poli bude mít zabezpečovací úlohu. Dále by pak měl mít možnost nastavit, kolik disků při simulaci nemá být funkčních a na kolik z nich se mají opravená data ukládat, protože už byly vyměněny.

2.2 Popis GUI rozšíření

Protože práce s příkazovým řádkem může být pro mnoho uživatelů velice náročná a výsledky nepřehledné, bylo k simulátoru zapotřebí také vyhovující grafické rozhraní. Tohoto úkolu se zhostil Bc. Kamil Jakubovič, který v rámci své bakalářské práce v roce 2017 [11] toto rozhraní vytvořil a nechal ho stejně jako DiskSim samotný volně ke stažení [12]. Uživateli pak



Obrázek 2.2: Grafické uživatelské rozhraní v internetovém prohlížeči

stačí v příkazové řádce spustit server, v libovolném internetovém prohlížeči zadat jeho adresu a spustí se grafické rozhraní simulátoru DiskSim.

2.2.1 Výběr parametrů simulace

Nejjednodušší způsob, jak v grafickém rozhraní spustit simulaci, je použít připravené scénáře. Většina konfiguračního souboru je v tomto případě již připravená a uživatel zde nastaví pouze několik málo parametrů, které však zásadně ovlivňují simulaci.

Uživatel si vybírá jaký typ diskového pole chce použít a v něm nastavuje ještě typ použitého disku, počet disků a velikost chunku [11]. Diskových polí zde může nakonfigurovat libovolný počet, přičemž každé z nich má své vlastní parametry.

Dalším důležitým bodem scénáře je nastavení generátoru zátěže. Uživatel si zde nastaví jaké vstupně výstupní operace se budou během simulace provádět. A opět má možnost nastavit více než jeden generátor s různými parametry.

Poslední věc kterou si uživatel může ve scénáři navolit je tzv. proměnný parametr simulace a podmínky pro ukončení simulace. Na obrázku 2.2 je ukázka grafického rozhraní během nastavování scénáře.

Pokud by chtěl uživatel v konfiguraci měnit jiné parametry než ty, které jsou nastavitelné ve scénáři, použije k tomu takzvané úlohy. Úlohy jsou

různé předem nastavené konfigurace simulací. Uživatel si může vybrat ze seznamu jednu konkrétní úlohu a tu pak pozměnit libovolně podle vlastních potřeb. Úlohy se skládají z částí. Těmi jsou jednotlivé bloky konfiguračního souboru DiskSimu. Uživatel si pak může zobrazovat a upravovat danou konfiguraci po těchto blocích.

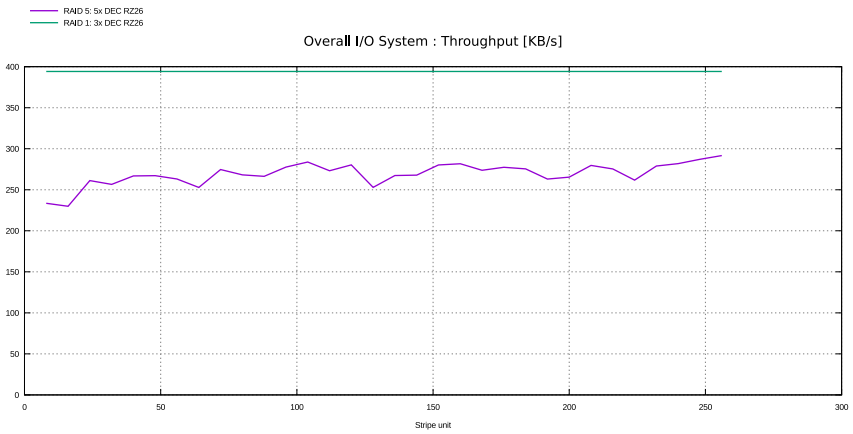
2.2.2 Zpracování simulací v cyklu pro různé parametry a grafický výstup

Před spuštěním simulace, ať už z připraveného scénáře nebo z nastavené úlohy, si může uživatel ještě vybrat jeden libovolný parametr a nastavit mu více hodnot. Hodnoty zadá buď jejich výčtem nebo rozsahem (s případnou velikostí skoku).

Pro každou ze zadaných hodnot tohoto parametru pak proběhne samostatná simulace. Výstupy těchto simulací se ukládají do oddělených adresářů. Z nich se pak vyberou všechny zapsané statistiky a ty se zobrazí v uživatelském rozhraní seskupené do několika logických skupin. [11]. Uživatel pak může jednoduše přepínat mezi výsledky jednotlivých simulací a vzájemně je porovnávat.

Kromě strukturovaného textového výstupu si však ještě uživatel může zobrazit i grafické znázornění. Pokud byl v konfiguraci zadán nějaký proměnný parametr, vygeneruje se pomocí programu gnuplot pro určité statistiky i graf. Na ose X jsou jednotlivé hodnoty proměnného parametru a na ose Y potom hodnoty daných statistik.

Uživatel má navíc možnost si takto zpracovaná výstupní data uložit. Uložit je může buď jako CSV soubor s daty nebo přímo již vygenerovaný graf a to ve vektorovém nebo rastrovém formátu. Obrázek 2.3 ukazuje jeden z vygenerovaných grafů v grafickém rozhraní.



Obrázek 2.3: Graf závislosti dané statistiky na hodnotách parametru

Kapitola 3

Analýza

Abych mohla simulátor rozšiřovat, bylo důležité nastudovat si jeho interní chování. Neexistuje k němu žádná celistvá technická dokumentace, takže jsem musela rozebírat přímo zdrojové kódy.

3.1 Popis simulace v simulátoru DiskSim

DiskSim funguje tak, že dostane požadavek na nějakou vstupně výstupní operaci (čtení nebo zápis dat) buď z předem zaznamenané činnosti nebo z generátoru. Tento požadavek pak prochází celou topologií a přitom se pečlivě zaznamenávají informace o jeho cestě. Z nich se pak vygenerují požadované statistiky.

3.1.1 Načítání konfigurace do datových struktur

Pro načítání parametrů z konfiguračního souboru používá DiskSim knihovnu libparam. Ta funguje nejen jako parser konfiguračního souboru, ale dokáže jednotlivým strukturám zadané parametry i přiřadit. K tomu však potřebuje pro každý modul speciální tzv. `modspec` soubor, ve kterém jsou definovány jeho parametry. Z tohoto souboru se pak při kompilaci vygeneruje zdrojový kód, který obstarává inicializaci parametrů.

Navíc se z něj může vygenerovat i dokumentace pro jednotlivé konfigurační parametry. Takto vygenerovanou dokumentaci pro modul `disksim_logorg` s nově přidanými konfiguračními parametry uvádím v příloze B.

Přiřazování načtených hodnot do parametrů datových struktur řídí modul `disksim_loadparams`. Poté co načte všechna zařízení a jejich topologii

zavolá funkci `iodriver_load_logorg` pro všechna disková pole, které v konfiguraci najde.

Z této funkce se zavolá `disksim_logorg_loadparams`, kde se vytváří struktura `disksim_logorg` do ní se pomocí rutiny `lp_loadparams` přiřadí všechny parametry. Tato rutina totiž volá funkce vygenerované z `modspec` souboru.

Nakonec se ještě z `disksim_loadparams` zavolají nastavovací funkce pro generátor zátěže, pokud byl v konfiguraci zadán.

Po načtení příslušných parametrů z konfigurace se ještě musí všechna zařízení inicializovat. O to se stará modul `disksim_iodriver`. Poté, co jsou hotové všechny řadiče, sběrnice a fyzické disky, zavolá ještě funkci `logorg_initialize`, která nastaví i ostatní potřebné parametry všem diskovým polím dané simulace.

3.1.2 Průběh simulace V/V operace

Zpracování každého požadavku na čtení nebo zápis má na starosti modul `disksim_iodriver`. V něm se požadavky zařazují do fronty a vyřízené se z ní vyřazují a také se tu zpracovávají statistiky pro daný požadavek.

Když ovladač vyjme z fronty požadavek, zavolá funkci `logorg_maprequest`. Tam se zjišťuje, do jakého pole by měl požadavek postoupit. Všechny požadavky musí projít přes nějaké diskové pole, jinak nemohou být dále zpracovávány. Diskové pole reprezentuje modul `disksim_logorg` a na něj napojený `disksim_redun`. Právě v těchto modulech se požadavky rozdělují na jednotlivé disky.

Ve funkci `logorg_maprequest` se rozhoduje, jakého typu RAIDu je používané diskové pole a podle toho se zavolá další funkce na práci s daným požadavkem.

V datové struktuře požadavku je uložena hodnota, kolik bloků dat se má přečíst nebo zapsat a kde tato operace začíná. Ve funkcích pro jednotlivé typy RAIDů se pak musí dopočítat, do kolika datových chunků daná operace zasáhne. Pro každý zpracovávaný chunk se vygeneruje nový požadavek, protože operace zasáhla na další disk. Tento požadavek má pak velikost zmenšenou o počet bloků, které se už zpracovaly na předchozím chunku.

V praxi to znamená, že se vytvoří tolik požadavků, přes kolik chunků operace zasáhla. Tyto požadavky se ihned po vytvoření zařadí do fronty požadavků a čekají na další zpracování.

Pokud daný RAID obsahuje nějaké redundantní disky a pokud by přišel požadavek na operaci zápisu, musí se vygenerovat ještě navíc další požadavky na zapsání zabezpečovacích dat. I tyto požadavky se pak zařadí na příslušné místo ve frontě.

Fronta s požadavky se tak během simulace stále mění, jak tam nové požadavky přibývají a vyřízené se odebírají. Simulace skončí, když se fronta celá vyprázdní.

3.1.3 Zpracování požadavku v diskovém poli typu RAID5

Diskové pole typu RAID5 může být obsluhováno různými funkcemi. Která z nich se zavolá záleží na kombinaci parametrů `Addressing mode` a `Distribution scheme`, které udávají, jak jsou jednotlivé chunky na discích rozmístěny. Nejpoužívanější kombinace těchto parametrů je:

- `Addressing mode = Parts`
- `Distribution scheme = Striped`

Ta seřadí chunky přes jednotlivé disky do redundantních sad, jak je ukázáno na obrázku 1.3. K tomu musí být vyplněný ještě parametr `Parity rotation type`, který udává směr rotace chunků. V DiskSimu je pro něj zatím implementována pouze jedna varianta rotace doleva.

Pokud uživatel zadá tuto kombinaci parametrů, budou se požadavky vyřizovat a přepočítávat pomocí tzv. paritní tabulky. Tato tabulka má počet sloupců roven počtu disků v poli a je v ní uložena informace o rozložení paritních chunků mezi jednotlivé disky. Protože se rozložení pořád opakuje, stačí aby měla i počet řádků roven počtu disků a případné číslo chunku se později dopočítá.

Hlavní funkcí, která pro tyto parametry zpracovává požadavky, je `logorg_parity_table`. Ta zpracovává bloky požadavku v cyklu po jednotlivých chuncích. Když narazí na konec redudndatní skupiny, přeskočí do další a pokud byl požadavek na operaci zápisu, přiřadí do fronty nový požadavek pro zápis parity. Až zbude v požadavku menší počet bloků než je velikost chunku, tak při operaci čtení se tento zbytek spojí s následujícím požadavkem.

Při požadavku na zápis se pak volí mezi různými strategiemi vypočítávání parity, když nebyla načtená data z celé redundantní skupiny. Volbu této strategie může uživatel ovlivnit nastavením parametru `RMW vs. reconstruct`. Ten udává hranici rozhodování, při jakém počtu načtených dat se nová parita bude počítat z nových dat a staré a nové parity a při jakém počtu je výhodnější dočíst zbylá data z redundantní sady a dopočítat paritu pomocí nich.

Vytvoření nových požadavků pro načtení dat pro počítání parity obstarávají zvláštní funkce, které se volají z `logorg_parity_table` podle nastavených podmínek. V hlavní funkci se pak ještě vytvoří požadavek na zapsání dopočítané parity.

3.2 Skripty pro spuštění DiskSimu s nastavenou konfigurací

Pokud by uživatel chtěl pracovat s DiskSimem přímo v příkazové řádce a ne přes grafické rozhraní, má k němu navíc k dispozici několik skriptů a šablon, které mu pomohou práci usnadnit. Ty k DiskSimu přidal Bc. Kamil Jakubovič [1].

Ve speciálním adresáři `script`, který je k DiskSimu přiložený se nachází tyto skripty:

- `disksim.sh` - spustí DiskSim se zjednodušenými parametry, které mohou být generovány z dalších skriptů
- `gen_raid.sh` - podle zadaných parametrů vygeneruje konfigurační soubor pro DiskSim s příslušným typem RAIDu
- `run_raid_procs.sh` - spustí DiskSim vygeneruje požadované statistiky buď v textové formátu nebo formou grafových obrázků

Další užitečnou pomůckou jsou připravené filtry statistik, které výrazně zjednodušují a zpřehledňují textový výstup samotného DiskSimu. Uživatel si pomocí nich může jednoduchým spuštěním zobrazit například pouze souhrnné statistiky, aby měl to nejdůležitější hned po ruce.

3.2.1 Šablony konfigurací

Důležitou součástí těchto skriptů jsou šablony konfigurací. V adresáři `script/templates` se nachází šablony, ze kterých lze poskládat všechny podporované RAIDy. Jednotlivé šablony obsahují části konfiguračního souboru, ze kterých se může sestavit kompletní konfigurace.

Hlavní je šablona `main.tmpl`, ve které se nachází specifikace všech zařízení, jejich topologie a výběr statistik. Dále jsou zde i části bloků pro konfiguraci RAIDů a generátoru zátěže, které obsahují společné parametry. V dalších šablonách jsou pak konfigurace odlišné pro jednotlivé typy RAIDů a generátory zátěže.

Aby bylo použití skriptů co nejjednodušší, nedá se při jejich spouštění nastavovat mnoho parametrů. Pokud by uživatel chtěl vygenerovat více pozměněnou konfiguraci, může si příslušné parametry najít ve správné šabloně a tam je změnit.

Kapitola 4

Návrh a realizace rozšíření

4.1 Zobecnění konfigurace diskových polí pomocí Reed-Solomonova schématu

Aby nedošlo ke ztrátě původní funkčnosti, neprovedla jsem změny v původních funkcích, které uskutečňovaly simulace pro RAID0 a RAID5. Vytvořila jsem nové funkce, které pomocí Reed-Solomonova schématu obstarávají RAID0, RAID5 nebo RAID6. Uživatel si pak v konfiguračním souboru bude moci vybrat, které funkce budou pro jeho RAIDy použity.

Pokud v konfiguračním souboru v parametru `Redundancy_scheme` uživatel zadá původní názvy RAIDů, které DiskSim nadefinoval, budou požity původní funkce. Ty však neumožňují práci v degradovaném režimu. Původní názvy jsou `Noredun` pro RAID0 a `Parity_rotated` pro RAID5 [4].

Pokud však uživatel v parametru `Redundancy scheme` zadá přímo řetězec `RAID0`, `RAID5` nebo `RAID6`, bude použita nová implementace. Zde všechny RAIDy obsluhuje pouze jedna obecná funkce. Ta pracuje se všemi RAIDy stejně, pouze na základě vybraného RAIDu zapisuje příslušný počet paritních dat.

Uživatel však může zadat ještě úplně novou hodnotu `RAIDRS` do parametru `Redundancy scheme`, která mu umožňuje zadat libovolný počet paritních disků. V takovém případě pak ještě musí vyplnit nový parametr `Parity_disks` a nastavit mu příslušnou hodnotu.

Pokud by uživatel zvolil konkrétní RAID a zároveň nastavil i parametr `Parity_disks`, bude tento parametr ignorován a počet paritních disků se nastaví podle typu RAIDu.

4.2 Simulace výpadku jednoho či více disků

Simulaci výpadku disků řídí uživatel. Pokud si přeje nasimulovat situaci, ve které je jeden nebo více disků nefunkčních, musí tento údaj zadat v konfiguračním souboru. K tomu slouží nový parametr `Failed disks` ve struktuře `logorg`. Jeho hodnotou je počet porouchaných disků.

Zde však platí omezení, že počet porouchaných disků nesmí přesáhnout počet disků paritních, jinak by nebylo možné poškozená data obnovit. Pro `RAID5` tedy může uživatel nastavit maximálně jeden poškozený disk a pro `RAID6` dva. Při obecném `RAIDRS` by si pak měl uživatel ohlídat, že parametr `Failed disks` není vyšší než parametr `Parity disks`. Tato omezení se testují při inicializaci parametrů a pokud by byla nějaká podmínka porušena, simulace neproběhne.

V simulovaném prostředí je jedno, který fyzický disk se porouchá, protože zabezpečovací data jsou distribuována na všechny disky. A proto simulátor nastavuje vždy příslušný počet poškozených disků od konce pole.

Práci v degradovaném režimu však umožňují pouze nově implementované funkce pro RAID5 a vyšší. Pokud by uživatel do konfiguračního souboru zadal názvy původních RAIDů, budou všechny nové parametry ignorovány a DiskSim bude pracovat se všemi disky jako s nepoškozenými.

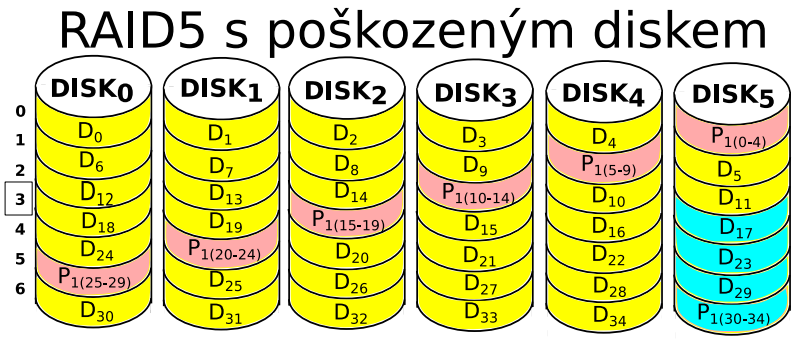
Pokud operace čtení padne na poškozený disk, musí nejprve dojít k rekonstrukci dat v daném chunku disku pomocí přečtení celé datové řady a vypočítání poškozených dat. Pokud je disk nahlášený jako vyměněný, tato opravená data se na něj pak rovnou zapíší a nastaví se příznak, že tento chunk už byl opraven.

Pokud operace zápisu padne na poškozený disk, který je hlášený jako vyměněný, tak se na něj data zapíší. Poté se zapíší i nová data na paritní disky. Pokud disk ještě vyměněný nebyl, zapíší se pouze nová paritní data.

Operace zápis paritních dat se generuje v simulaci jako nový požadavek na zápis, takže pokud bude poškozený disk, na který se měla parita zapsat, zapíše se pouze v případě, že disk byl vyměněný.

4.3 Simulace rekonstrukce jednoho či více disků

Za požadavek k rekonstrukci dat simulátor považuje požadavek čtení celé jedné redundantní skupiny, což znamená čtení všech jejích datových chanků. Pokud takový požadavek přijde, simulátor nejen přečte všechna tato data, ale ještě přidá požadavek na zápis nových dat na vyměněné poškozené disky.



Obrázek 4.1: Strategie použití fence při rekonstrukci dat - fence = 3

Uživatel se může rozhodnout, že chce nasimulovat rekonstrukci celého diskového pole. Toho dosáhne tak, že simulátoru pošle požadavek na sekvenční čtení celého pole v blocích odpovídající velikosti redundantní skupiny. Toho dosáhne správným nastavením vestavěného simulátoru zátěže.

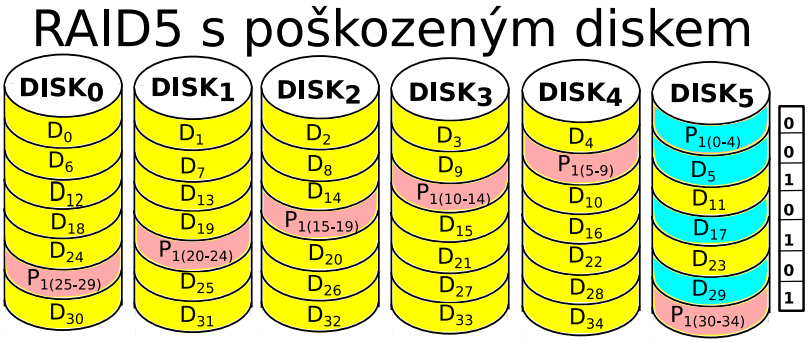
Další možností je nechat simulátor pracovat v degradovaném režimu. V takovém případě simulátor musí sám poznat kdy pracuje s poškozeným diskem a data rekonstruuje až při potřebě jejich použití. V takovém případě má uživatel také možnost už v počáteční konfiguraci nastavit procento opravení poškozených disků pomocí parametru `Repaired part`.

Pak ovšem nastává potřeba uchovávat v simulátoru informace o tom, které části poškozených disků už byly opraveny, aby zbytečně nedocházelo k rekonstrukci už jednou opravených dat.

4.3.1 Strategie použití fence nebo bitmapy pro rekonstrukci

Aby bylo možné, opravovat poškozené disky po částech, musí simulátor nějak uchovávat informaci o tom, které části disku už byly opraveny. Zde si může uživatel vybrat ze dvou možností pomocí parametru `Repair type`. Tento parametr může nastavit buď na hodnotu 1 nebo 2.

1 znamená, že se informace o opravených částech disku budou ukládat pomocí tzv. `fence`, což by se dalo přeložit jako plot. Představme si, že disk má chunky sekvenčně očíslované jako je to na obrázku 4.1. Fance je číslo od 0 do počtu chunků na jednom disku -1, které udává první ještě neopravený chunk. V praxi to znamená, že chunky s nižším číslem, než je hodnota fence, se už považují za opravené a obsahují správná data. Zbylé chunky ještě čekají na opravení.



Obrázek 4.2: Bitová mapa pro rekonstrukci dat - bitmap = {0,0,1,0,1,0,1}

Pokud by přišel požadavek čtení na opravovaném disku do chunku, který má stejné číslo jako je hodnota fence, data na tomto chunku se zrekonstruují a zapíší a fence se zvýší o 1. Pokud by přišel požadavek do chunku s vyšším číslem, data se zrekonstruují, ale nezapíší a hodnota fence se nemění.

Podobně je tomu i s požadavkem na zápis. Pokud zapisuji do opravené části disku, vše probíhá jako na zdravém zařízení. Pokud zapisuji na chunk s číslem stejným jako fence, zapíšu tam data a zvýším fence. Pokud by zápis přišel na chunk s vyšším číslem, nezapisují se na něj data vůbec.

Hodnota 2 v `Repair type` zase znamená, že se data budou rekonstruovat při každém požadavku na čtení nebo zápis na základě informace uložené v bitové mapě. Jednotlivé bity zde reprezentují jednotlivé chunky disku. 0 znamená že chunk ještě nebyl opraven a 1 že ano, jak ukazuje obrázek 4.2.

Při požadavku na čtení se pak podle hodnoty příslušného bitu buď data přečtou jako ze zdravého disku a nebo zrekonstruují, zapíší a bit se přehodí na hodnotu 1. Při požadavku na zápis se pak data vždy zapíší a pokud byl bit nulový, také se mu nastaví hodnota 1.

Pokud by bylo poškozených a vyměněných disků více, má každý takový disk svůj vlastní fence nebo bitmapu. Při rekonstrukci dat se pak rekonstruuje celá redundantní skupina a zapisují se obnovená data a nastavují se příznaky opravení na všech vyměněných discích.

4.3.2 Generování zátěže rekonstrukce

Uživatel má i možnost nasimulovat samostatně rekonstrukci celého diskového pole najednou. K tomu se může použít vestavěný generátor zátěže. Všechny jeho parametry se musí nastavit v konfiguračním souboru. Jak se celý generátor správně nastavuje se můžeme dočíst v dokumentaci DiskSimu [4].

Já zde uvedu pouze parametry, ke kterým musí uživatel zadat specifické hodnoty, aby nasimuloval rekonstrukci celého diskového pole. Tyto parametry jsou:

- `Number of I/O requests to generate = x` - x je počet chnuků na jednom disku
- `Generators = uniform` - typ generátoru zátěže
- `Blocking factor = y` - y je velikos celé jedné datové řady diskového pole - velikost jednoho chunku * počet datových disků
- `Probability of sequential access = 1` - zadává se sekvenční přístup
- `Probability of local access = 0`
- `Probability of read access = 1` - zadávají se požadavky pouze pro čtení
- `Probability of time-critical request = 1` - tyto požadavky by měly mít přednost

4.4 Změny simulace v simulátoru DiskSim

4.4.1 Rozšíření datových struktur

Ze všeho nejdříve bylo nutné přidat několik nových položek do struktury `logorg`. Nejdůležitější je položka `int parity_disks`, která udává počet paritních disků v Reed-Solomonově schématu.

Pomocí dalších položek uživatel nastavuje parametry pro práci v degradovaném režimu a rekonstrukci dat. Tyto parametry jsou:

- `int psrity_disks` - počet paritních disků
- `int failed_disks` - počet vadných disků
- `int repairing_disks` - počet vyměněných disků, na kterých se data mohou rekonstruovat
- `int repair_type` - typ rekonstrukce - zda se bude rekonstrukce provádět pomocí Fance nebo Bitmapy
- `double repaired` - čílo od 0 do 1 - udává procento již opravených chybných dat

Hodnoty všech těchto položek nastavuje uživatel pomocí daných parametrů v konfiguračním souboru.

Dalšími položkami, které bylo třeba přidat, byly pole pro uchování informace o opravených částech poškozených disků. Jsou to `int *fance` a `int *bitmap`. Na základě hodnoty uložené v `int repair_type` se pak rozhoduje, která položka se bude používat při rekonstrukcích dat.

4.4.2 Dosavadní změny v algoritmech a průběhu simulace

Ze všeho nejdříve jsem musela upravit funkci `logorg_initialize`. Přidala jsem do ní kontroly nových konfiguračních parametrů a pomocí nich pak ještě přepočítám kolik chunků v redudndatní skupině je datových a kolik se na ně vejde bloků dat. V další části této funkce jsem pak ještě musela naalokovat místo pro pole jednotlivých fence nebo bitových map.

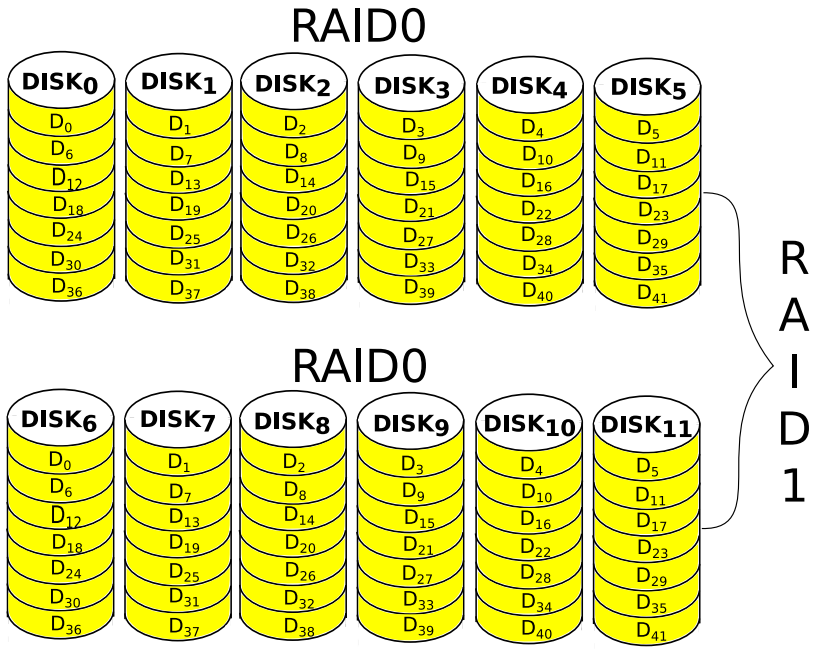
Poté jsem vytvořila nové funkce pro mapování požadavků. Hlavní funkce, která se volá z `logorg_maprequest`, se nazývá `logorg_reed_solomon`. Její kód vychází z úpravy kódu funkce `logorg_parity_table`, která v původní verzi mapovala požadavky pro pole typu RAID5 pomocí paritní tabulky.

Proto jsem musela vytvořit i funkce `logorg_parity_table_rs` a `logorg_create_table_left_sym_rs`, ktaré tuto tabulku vytváří a vyplňují. Ty jem upravila tak, aby se všechna paritní data zapisovala do tabulky na správné chunky a celá redundantní skupina rotovala. Pokud diskové pole neobsahuje žádný paritní disk, tak se chunky nerotují, aby se dalo toto schéma nazvat RAID1.

Hlavní funkci jsem prozatím upravila do podoby, ve které je schopná správně obsluhovat požadavky na čtení pro všechny nově zadané RAIDy, pokud jsou všechny disky funkční. K tomu se používá parametr `partsperstripe`, ketrý udává počet datových chunků v jedné redundantní skupině. Ten se ve funkci `logorg_initialize` nově počítá jako rozdíl celkového počtu disků a paritních disků. Při zpracovávání požadavků pak slouží jako hranice, kdy se data začínají načítat z další redundantní sady.

4.5 Vnořovaní RAIDů

Dalším úkolem mé práce mělo být předělání simulátoru tak, aby bylo možné RAIDy zřetězit a vytvořit tak např. RAID1+0 nebo RAID0+1, který ukazuje obrázek 4.3. Simulátor je však nyní nastaven tak, že v konfiguraci se v modulu `disksim_logorg` do parameru `devices`, který popisuje z čeho



Obrázek 4.3: RAID0 zrdcadlený pomocí RAIDU1

se diskové pole skládá, můžou zadávat pouze koncová disková zařízení. Není tedy možné, aby uživatel v konfiguraci ibovolně vnořoval jakékoli RAIDy dohromady.

Aby se toto omezenení zrušilo, musel by se předělat kód pro načítání parametrů na několika místech. Nejprve by se musely zrušit podmínky, které udávají, že v poli smí být pouze disky. Dále, při načítání zařízení do pole, by se musel prohledávat nejen seznam diskových zařízení, ale i seznam už načtených diskových polí.

Pokud by se místo disku do vrnícho RAIDU uložilo jiné diskové pole, musela by se spočítat jeho celková velikost. V parametrech pro ukládání informací o veliostech jednotlivých disků by pak byly uloženy velikosti celých RAIDů. Aby se však Vnitřní pole mohly stát součástí vnějšího, musí být v konfiguračním souboru uvedeny první.

Do konfigurace bychom ještě mohli přidat nový parametr s názvem `Logorgs`, do kterého by se zapisovala hodnota 0 nebo 1 podle toho, zda bude mít toto pole místo disků v parametru `devices` jiná disková pole.

Podle tohoto příznaku, by pak požadavky, které vznikly v `logorg_parity_table`, ve skutečnosti nebyly požadavky na chunky disku, ale na jednotlivé disky vnitřního diskového pole. Každý nový požadavek by se musel zkopirovat a zavolala by se na pro něj funkce

`logorg_maprequest`. Ta by přemapovala tyto požadavky na jednotlivé disky vnitřních polí.

Pokud bychom toto provedli i ve funkci `logorg_shadowed` pro RAID1, mohli bychom vnořovat RAID0, RAID5, RAID6 a RAIDRS do RAID1 i obráceně.

Tuto funkcionalitu jsem však z časových důvodů do DiskSimu nedoimplementovala, takže zatím zůstala pouze ve formě návrhu.

Kapitola 5

Testování

Během implementvání jsem spouštěla různé simulace, abych zjistila, zda moje funkce pro RAID0 a RAID5 pracují stejně, jako ty původní. Vzhledem k tomu, že kód pro mapování Reed-Solomonova schématu vychází z původního kódu pro RAID5, není překvapením, že při stejně nastavené konfiguraci dostáváme stejné statistiky.

Pro RAID0 už ovšem simulátor pracuje jinak než původní funkce. Překvapilo mě, že pro stejnou konfiguraci vychází statistiky zpracované pomocí Reed-Solomonova schématu hůře, než ty z původní implementace. Zpracovat stejné množství požadavků trvá v nové implementaci déle, jak ukazují tyto souhrnné statistiky:

- `Overall I/O System Requests per second` - nová implementace: 66.871461
- `Overall I/O System Requests per second` - původní implementace: 67.004155
- `System logorg #0 Completely idle time` - nová implementace: 614239.367166
- `System logorg #0 Completely idle time` - původní implementace: 614239.367166

Závěr

Cílem teoretické části práce byla analýza kódu simulátoru DiskSim se zaměřením na disková pole. Detailně jsem v této části popsala jak DiskSim simulace provádí a jaký způsobem ho může uživatel nastavovat.

V praktické části, zase bylo cílem navrhnout a implementovat rozšíření simulátoru o práci s daty v degradovaném režimu a možnost vygenerevání zátěže pro simulace rekostrukce disku. Data by v tomto případě měla být zabezpečena pomocí Reed-Solomonova schématu.

Navrhla jsem proto, jak by měla být pozměněna konfigurace DiskSimu a doimlementovala načítání potřebných parametrů do přidaných datových struktur. Uživatel si nyní v konfiguraci může vybrat, zda chce simulace provádět nad klasickými poli jednotlivých typů RAIDů, nebo zda se má pro ukládání a zabezpečení dat použít Reed-Solomonovo scéma.

V další části návrhu jsem pak detailně popsala, jakým způsobem by měla probíhat simulace, pokud by se v poli vyskytly nefunkční disky. Vysvětlila jsem zde, jakým způsobem má proběhnout simulace rekonstrukce dat z poškozeného disku a jakým způsobem se bude ukládat informace o již opravených částech.

Poslední částí mé práce je implementace navržených postupů zpracování požadavků v simulaci. Na tuto část mi bohužel nezbyl dostatek času, a tak se mi ji podařilo splnit pouze částečně.

Funkce, která zpracovává požadavky pole typu obecného Reed-Solomonova schématu zatím dokáže zpracovávat ve všech konfiguracích pouze požadavky na čtení, pokud pole neobsahuje žádný vadný disk. A pro konfiguraci typu RAID0 a RAID5 může zpracovávat i zápisy.

Zbylou implementaci pro simulace operací nad nefunkčními disky mám samozřejmě v plánu co nejdříve dokončit.

Literatura

[1] Jakubovič, K.: *Disksim builder and scripts*[online]. květen 2017, [cit. 2018-10-29]. Dostupné z: `https://gitlab.fit.cvut.cz/jakubkam/disksim`

[2] Robinson, S.: *Simulation: The Practice of Model Development and Use*. John Wiley & Sons, 2004, ISBN 0470847727.

[3] Law, A.; Kelton, W.: *Simulation Modeling and Analysis*. Boston: McGraw-Hill Publishing, třetí vydání, 2000, ISBN 0070592926.

[4] Bucy, J. S. S. W. a., J. S.; Schindler: *The DiskSim Simulation Environment Version 4.0 Reference Manual*. květen 2008. Dostupné z: `http://www.pdl.cmu.edu/PDL-FTP/DriveChar/CMU-PDL-08-101.pdf`

[5] Ganger, G.: *The DiskSim Simulation Environment (v4.0). The Parallel Data Lab*[online]. květen 2015, [cit. 2018-10-31]. Dostupné z: `http://pdl.cmu.edu/DiskSim/index.shtml`

[6] Microsoft: *SSD Extension for DiskSim Simulation Environment*[online]. březen 2009, [cit. 2018-10-31]. Dostupné z: `http://research.microsoft.com/research/downloads/Details/b41019e2-1d2b-44d8-b512-ba35ab814cd4/Details.aspx`

[7] Patterson, G. a. K. R. H., A. D. a Gibson: A Case for Redundant Arrays of Inexpensive Disks (RAID). *1988 ACM Conference on Management of Data*, červen 1988: s. 109–116.

[8] Plank, J. S.: A Tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-like Systems. Technická Zpráva CS-96-332, University of

Tennessee, červenec 1996, [cit. 2018-12-23]. Dostupné z: http://web.eecs.utk.edu/~plank/plank/papers/CS-96-332.html

[9] Reed, G., I. a Solomon: Polynomial Codes Over Certain Finite Fields. *Journal of the Society for Industrial and Applied Mathematics*, ročník 8, č. 2, 1960. Dostupné z: https://doi.org/10.1137/0108018

[10] Geisel, W. A.: *Tutorial on Reed-Solomon Error Correction Coding*[online]. Lyndon B. Johnson Space Center, Houston, Texas, srpen 1990, [cit. 2018-12-23]. Dostupné z: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19900019023.pdf

[11] Jakubovič, K.: *Grafické rozhraní simulátoru disků DiskSim*. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, květen 2017, bakalářská práce.

[12] Jakubovič, K.: *Disksim User Interface*[online]. červen 2017, [cit. 2018-12-23]. Dostupné z: https://gitlab.fit.cvut.cz/jakubkam/disksimui

Příloha A

Konfigurační soubor simulátoru DiskSim

```
disksim_global Global {
  Init Seed = 42,
  Real Seed = 42,
  Detailed execution trace = ../.work/execdetail,
  Output file for trace of I/O requests simulated
= ../.work/diskrequest,
  Stat definition file = statdefs
}

disksim_stats Stats {
  iodriver stats = disksim_iodriver_stats {
    Print driver size stats = 0,
    Print driver locality stats = 0,
    Print driver blocking stats = 0,
    Print driver interference stats = 0,
    Print driver queue stats = 0,
    Print driver crit stats = 0,
    Print driver idle stats = 0,
    Print driver intarr stats = 0,
    Print driver streak stats = 0,
    Print driver stamp stats = 0,
    Print driver per-device stats = 0
  },
```

```
  bus stats = disksim_bus_stats {
    Print bus idle stats = 0,
    Print bus arbwait stats = 0
  },
  ctlr stats = disksim_ctlr_stats {
    Print controller cache stats = 0,
    Print controller size stats = 0,
    Print controller locality stats = 0,
    Print controller blocking stats = 0,
    Print controller interference stats = 0,
    Print controller queue stats = 0,
    Print controller crit stats = 0,
    Print controller idle stats = 0,
    Print controller intarr stats = 0,
    Print controller streak stats = 0,
    Print controller stamp stats = 0,
    Print controller per-device stats = 0
  },
  device stats = disksim_device_stats {
    Print device queue stats = 1,
    Print device crit stats = 0,
    Print device idle stats = 0,
    Print device intarr stats = 0,
    Print device size stats = 1,
    Print device seek stats = 0,
    Print device latency stats = 0,
    Print device xfer stats = 0,
    Print device acctime stats = 0,
    Print device interfere stats = 0,
    Print device buffer stats = 0
  },
  process flow stats = disksim_pf_stats {
    Print per-process stats = 0,
    Print per-CPU stats = 0,
    Print all interrupt stats = 0,
    Print sleep stats = 0
  }
} # end of stats block
```

```
disksim_iodriver DRIVER0 {
  type = 1,
  Constant access time = 0.0,
  Use queueing in subsystem = 1,
  Scheduler = disksim_ioqueue {
    Scheduling policy = 3,
    Cylinder mapping strategy = 1,
    Write initiation delay = 0.0,
    Read initiation delay = 0.0,
    Sequential stream scheme = 0,
    Maximum concat size = 128,
    Overlapping request scheme = 0,
    Sequential stream diff maximum = 0,
    Scheduling timeout scheme = 0,
    Timeout time/weight = 6,
    Timeout scheduling = 4,
    Scheduling priority scheme = 0,
    Priority scheduling = 4
  } # end of Scheduler
} # end of DRV0 spec

disksim_bus BUS0 {
  type = 1,
  Arbitration type = 1,
  Arbitration time = 0.0,
  Read block transfer time = 0.0,
  Write block transfer time = 0.0,
  Print stats =  0
} # end of BUS0 spec

disksim_bus BUS1 {
  type = 1,
  Arbitration type = 1,
  Arbitration time = 0.0,
  Read block transfer time = 0.0512,
  Write block transfer time = 0.0512,
  Print stats =  1
} # end of BUS1 spec
```

```
disksim_ctlr CTLR0 {
  type = 1,
  Scale for delays = 0.0,
  Bulk sector transfer time = 0.0,
  Maximum queue length = 0,
  Print stats =  1
} # end of CTLR0 spec

source ./atlas10k.diskspecs

# component instantiation
instantiate [ statfoo ] as Stats
instantiate [ bus0 ] as BUS0
instantiate [ bus1 ] as BUS1
instantiate [ disk0 .. disk5 ] as QUANTUM_TORNADO_validate
instantiate [ driver0 ] as DRIVER0
instantiate [ ctlr0 ] as CTLR0

# system topology
topology disksim_iodriver driver0 [
  disksim_bus bus0 [
    disksim_ctlr ctlr0 [
      disksim_bus bus1 [
        disksim_disk disk0 [], disksim_disk disk1 [],
        disksim_disk disk2 [], disksim_disk disk3 [],
        disksim_disk disk4 [], disksim_disk disk5 []
      ]
    ]
  ]
]

# RAID5
disksim_logorg raid5 {
  Addressing mode = Array,
  Distribution scheme = Striped,
  Redundancy scheme = RAID6,
  Stripe unit  =  64,
  Number of copies =  0,
  Parity stripe unit =  64,
  Parity rotation type =  1,
  devices = [ disk0, disk1, disk2, disk3, disk4, disk5 ],
```

```
  Parity disks = 1,
  Failed disks = 1,
  Repairing disks = 1,
  Repair type = 1,
  Repaired part = 0.6,
  Copy choice on read =  6,
  Synch writes for safety =  0,
  RMW vs. reconstruct =  0.5,
  Time stamp interval =  0.000000,
  Time stamp start time =  60000.000000,
  Time stamp stop time =  10000000000.000000,
  Time stamp file name =  ../.work/stamps
}

# process flow spec
disksim_pf Proc {
  Number of processors =  1,
  Process-Flow Time Scale =  1.0
} # end of process flow spec

# synthetic workload spec
disksim_synthio Synthio {
  Number of I/O requests to generate =  5000,
  Maximum time of trace generated  =  100000.0,
  System call/return with each request =  0,
  Think time from call to request =  0.0,
  Think time from request to return =  0.0,
  Generators = [
    disksim_synthgen {
      Storage capacity per device = 2000000,
      devices = [ raid5 ],
      Blocking factor = 8,
      Probability of sequential access = 0.5,
      Probability of local access = 0.0,
      Probability of read access = 0.5,
      Probability of time-critical request = 1.0,
      Probability of time-limited request = 0.0,
      Time-limited think times = [ normal, 30.0, 100.0  ],
      General inter-arrival times = [ exponential, 0.0, 25.0  ],
      Sequential inter-arrival times = [ normal, 0.0, 0.0  ],
      Local inter-arrival times = [ exponential, 0.0, 0.0  ],
```

```
      Local distances = [ normal, 0.0, 40000.0  ],
      Sizes = [ exponential, 0.0, 8.0 ]
    }
  ] # end of generator list
} # end of synthetic workload spec
```

Příloha **B**

Dokumentace vygenerovaná ze souboru logorg.modspec

disksim_logorg	Addressing mode	string	required
This specifies how the logical data organization is addressed. `Array` indicates that there is a single logical device number for the entire logical organization. `Parts` indicates that back-end storage devices are addressed as though there were no logical organization, and requests are re-mapped appropriately.			

disksim_logorg	Distribution scheme	string	required
This specifies the data distribution scheme (which is orthogonal to the redundancy scheme). `Asis` indicates that no re-mapping occurs. `Striped` indicates that data are striped over the organization members. `Random` indicates that a random disk is selected for each request. N.B.: This is only to be used with constant access-time disks for load-balancing experiments. `Ideal` indicates that an idealized data distribution (from a load balancing perspective) should be simulated by assigning requests to disks in a round-robin fashion. Note that the last two schemes do not model real data layouts. In particular, two requests to the same block will often be sent to different devices. However, these data distribution schemes are useful for investigating various load balancing techniques [**?**]. N.B.: This is only to be used with constant access-time disks for load-balancing experiments.			

disksim_logorg	Redundancy scheme	string	required
This specifies the redundancy scheme (which is orthogonal to the data distribution scheme). Noredun indicates that no redundancy is employed. Shadowed indicates that one or more replicas of each data disk are maintained. Parity_disk indicates that one parity disk is maintained to protect the data of the other organization members. Parity_rotated indicates that one disk's worth of data (spread out across all disks) are dedicated to holding parity information that protects the other N-1 disks' worth of data in an N-disk organization. RAID0 indicates that no redundancy is employed. RAID5 indicates that one disk's worth of data (spread out across all disks) are dedicated to holding parity information that protects the other N-1 disks' worth of data in an N-disk organization. RAID6 indicates that two disk's worth of data (spread out across all disks) are dedicated to holding P and Q syndroms that protects the other N-2 disks' worth of data in an N-disk organization. RAIDRS indicates that M disk's worth of data (spread out across all disks) are dedicated to holding from P1 to PM syndroms that protects the other N-M disks' worth of data in an N-disk organization.			

disksim_logorg	Components	string	optional
This specifies whether the data organization's component members are entire disks (Whole) or partial disks (Partial). Only the former option is supported in the first released version of DiskSim.			

disksim_logorg	devices	list	required
List of device names to be included in this logical organization.			

disksim_logorg	Stripe unit	int	required
This specifies the stripe unit size. 0 indicates fine-grained striping (e.g., bit or byte striping), wherein all data disks in the logical organization contain an equal fraction of every addressable data unit.			

disksim_logorg	Synch writes for safety	int	required
This specifies whether or not an explicit effort should be made to do the N+1 writes of a parity-protected logical organization at "the same time" when handling a front-end write request with the read-modify-write (RMW) approach to parity computation. If true (1), then all reading of old values (for computing updated parity values) must be completed before the set of back-end writes is issued. If false (0), then each back-end write is issued immediately after the corresponding read completes (perhaps offering improved performance).			

disksim_logorg	Number of copies	int	required
This specifies the number of copies of each data disk if the logical organization employs Shadowed redundancy. Or this specifies the number of copies of all RAID if the logical organization employs RAID0, RAID5, RAID6 or RAIDRS redundancy. Otherwise, this parameter is ignored.			

disksim_logorg	Copy choice on read	int	required
This specifies the policy used for selecting which disk from a set of Shadowed replicas should service a given read request since any of them can potentially do so. 1 indicates that all read requests are sent to a single primary replica. 2 indicates that one of the replicas should be randomly selected for each read request. 3 indicates that requests should be assigned to replicas in a round-robin fashion. 4 indicates that the replica that would incur the shortest seek distance should be selected and ties are broken by random selection. 5 indicates that the replica that has the shortest request queue should be selected and ties are broken by random selection. 6 indicates that the replica that has the shortest request queue should be selected and ties are broken by policy 4 (see above). This parameter is ignored if Shadowed replication is not chosen.			

disksim_logorg	RMW vs. reconstruct	float	required
This specifies the breakpoint in selecting Read-Modify-Write (RMW) parity updates (verses complete reconstruction) as the fraction of data disks that are updated. If the number of disks updated by the front-end write request is smaller than the breakpoint, then the RMW of the "old" data, "old" parity, and "new" data is used to compute the new parity. Otherwise, the unmodified data in the affected stripe are read from the corresponding data disks and combined with the new data to calculate the new parity. This parameter is ignored unless some form of parity-based replication is chosen.			

disksim_logorg	Parity stripe unit	int	required
This specifies the stripe unit size used for the Parity_rotated redundancy scheme. This parameter is ignored for other schemes. The parity stripe unit size does not have to be equal to the stripe unit size, but one must be a multiple of the other. Use of non-equal stripe unit sizes for data and parity has not been thoroughly tested in the current release of DiskSim.			

disksim_logorg	Parity rotation type	int	required
This specifies how parity is rotated among the disks of the logical organization. The four options, as described in [?], are 1 - left symmetric, 2 - left asymmetric, This parameter is ignored unless Parity_rotated, RAID0, RAID5, RAID6 or RAIDRS redundancy is chosen.			

disksim_logorg	Parity disks	int	optional
This parameter specifies number of parity disks. This parameter is ignored unless RAID0, RAID5, RAID6 or RAIDRS redundancy is chosen.			

disksim_logorg	Failed disks	int	optional
This parameter specifies number of failed disks. This parameter is ignored unless RAID0, RAID5, RAID6 or RAIDRS redundancy is chosen.			

disksim_logorg	Repairing disks	int	optional
This parameter specifies number of disks just being reconstructed. This parameter is ignored unless RAID0, RAID5, RAID6 or RAIDRS redundancy is chosen.			

disksim_logorg	Repair type	int	optional
This specifies type od reconstruction scheme. Fence indicates that disk i reconstructed sequentially from LBN 0 to max. Bitmap indicates that disk i reconstructed during each read or write operations and recostructed ranges are marked in the bitmap.			

disksim_logorg	Repaired part	float	optional
This specifies how many percent of the data is corrected. This parameter is ignored unless RAID0, RAID5, RAID6 or RAIDRS redundancy is chosen.			

disksim_logorg	Time stamp interval	float	optional
This specifies the interval between “time stamps.” A value of 0.0 for this parameter disables the time stamp mechanism.			

disksim_logorg	Time stamp start time	float	optional
This specifies the simulated time (relative to the beginning of the simulation) of the first time stamp.			

disksim_logorg	Time stamp stop time	float	optional
This specifies the simulated time (relative to the beginning of the simulation) of the last time stamp.			

disksim_logorg	Time stamp file name	string	optional
This specifies the name of the output file to contain a log of the instantaneous queue lengths of each of the organization’s back-end devices at each time stamp. Each line of the output file corresponds to a single time stamp and contains the queue lengths of each device separated by white space. A value of “0” or of “null” disables this feature (as does disabling the time stamp mechanism).			

Příloha C

Seznam použitých zkratek a pojmů

DiskSim , název diskového simulátoru

RAID *Redundant Array of Inexpensive/Independent Disks*, vícenásobné diskové pole nezávislých disků

Reed-Solomonovo schéma , schéma pro zabezpečování dat v diskovém poli

POSIX *Portable Operating System Interface*, standard pro operační systémy

SSD *Solid State Disk*, disk bez pohyblivých částí, data ukládá do integrovaných obvodů

chunk souvislý úsek datových bloků na disku v diskovém poli

parita redundnatní data používaná pro opravování chyb

CLI *Comand Line Interface*, uživatelské rozhraní, kde uživatel zadává příkazy do příkazového řádku

GUI *Graphical User Interface*, grafické uživatelské rozhraní

CSV *Comma-Separated Values*, formát jednoduchécho souboru s tabulkovými daty, ve kterém jsou hodnoty odděleny čárkami

fence , hranice na vyměněném disku mezi opravenými a neopravenými daty

alokace , zarezervování místa paměti pro danou datovou strukturu

Příloha D

Obsah přiloženého DVD

- `README.txt` ... stručný popis obsahu CD
- `BP_Beruskova_Jana_2019.pdf` ... text práce ve formátu PDF
- `thesis` ... zdrojová forma práce ve formátu LaTeX
  - `img` ... obrázky použité v textu práce
- `DiskSim_4.0/` ... adresář se všemi komponentami DiskSimu
  - `Makefile` ... Makefile pro zkompilování DiskSimu
  - `bin` ... adresář se spustitelnými soubory DiskSimu
  - `build` ... adresář pro zdrojové soubory
    - `disksim-4.0` ... adresář pro zdrojové soubory DiskSimu
      - `Makefile` ... Makefile pro zkompilování původního DiskSimu
      - `src` ... adresář pro zdrojové kódy DiskSimu
        - `modules` . adresář pro modspec soubory a generované kódy
      - `valid` ... adresář pro testovací konfigurační soubory
      - `doc` ... adresář pro dokumentaci DiskSimu
        - `disksim4-manual.pdf` ... uživatelská příručka DiskSimu
      - `...` ... další adresáře zdrojové adresáře a soubory
  - `src` ... adresář s původní verzí DiskSimu a přídavných patchů
  - `patches` ... adresář pro patch soubory pro původní DidkSim
  - `script` ... adresář se skripty které zjednodušují spouštění DiskSimu