ITAT 2013 Proceedings, CEUR Workshop Proceedings Vol. 1003, pp. 69–74
http://ceur-ws.org/Vol-1003, Series ISSN 1613-0073, © 2013 M. Čermák, F. Zavoral

Dosiahnutie vysokej dostupnosti v D-Boboxe

Miroslav Čermák a Filip Zavoral

Univerzita Karlova v Praze, Česká Republika
{cermak, zavoral}@ksi.mff.cuni.cz

Abstrakt: Prúdové spracovanie veľkých dát v distribuovanom prostredí prináša radu rôznych výziev. Jednou z nich je dosiahnutie vysokej dostupnosti (high availability), čo predstavuje zabezpečenie odolnosti distribuovaného výpočtu proti výpadku uzlov z dôvodu hardvérovej, či softvérovej chyby, alebo výpadku spojenia medzi uzlami. Výpadky sú problematické, pretože ich následkom dochádza k narušeniu výpočtu – strate časti medzivýsledkov a strate stavu havarovaného uzlu. To je vo väčšine prípadov neakceptovateľné a je nutné začať spracovanie od začiatku, čo predlžuje výslednú dobu spracovania a spotrebováva hardvérové prostriedky. V tomto článku zhrnieme súčasný stav poznania v oblasti dosiahnutia vysokej spoľahlivosti pri prúdovom spracovaní dát a navrhneme spôsob rozšírenia systému D-Bobox o podporu vysokej dostupnosti.

1 Úvod

V poslednom čase je venovaná veľká pozornosť novej triede aplikácií - systémom pre spracovanie prúdových dát (stream processing systems - SPS) [1, 2, 3], ktoré musia spracovávať obrovské objemy neustále prúdiacich dát s nízkou latenciou. Medzi takéto systémy môžeme zaradiť napr. spracovanie finančných dát v reálnom čase, monitorovanie pacientov pomocou rôznych senzorov, sledovanie a vyhodnocovanie stavu dopravy atď. Prúdové spracovanie sa ukazuje ako efektívny nástroj nielen pre spracovanie priebežne generovaných dát, ale aj veľkých statických dát, ako napríklad sémantické databázy [4]. Zdroje dát SPS systémov a spracovanie jednotlivých prúdov dát môžu byť distribuované, čo poskytuje priestor na škálovanie ich výkonu. Takéto systémy nazveme distribuované SPS - DSPS.

Typickým postupom zvýšenia výkonu v distribuovanom systéme prúdového spracovania dát je pridanie ďalších serverov. To avšak zvyšuje riziko výpadku, ktorý negatívne ovplyvňuje výkon a spoľahlivosť celého distribuovaného systému, pretože mnoho systémov požaduje presné (databáze) a pravidelné výsledky (vyhodnocovanie finančných operácií, sledovanie pacientov). V efektívnom systéme sa preto stáva nevyhnutná prítomnosť komponenty pre vysokú dostupnosť (high availability - HA), ktorá umožňuje rýchle a korektné obnovenie a má minimálny dopad na bezvýpadkové spracovanie.

K dosiahnutiu HA v distribuovaných systémoch prúdového spracovania dát je nutné poskytnúť sadu algoritmov, ktoré vykonávajú nasledujúce úlohy:

1. periodická a inkrementálna záloha (alebo replikácia) stavu uzlov, na ktorých prebieha výpočet
2. detekcia chyby
3. výber uzlu ktorý preberie úlohy havarovaného
4. obnovenie strateného stavu po výpadku
5. zvládnuť rozdelenie siete

V tejto práci najprv v kapitole 2 definujeme typy zotavenia podľa [5], následne v kapitole 3 predstavíme základné HA metódy zamerané primárne na body (1) a (4). Následne predstavíme systém D-Bobox 4 a v kapitole 5 možnosti nasadenia HA v ňom.

2 Typy zotavenia

Základná požiadavka kladená na algoritmy zotavenia je schopnosť maskovať výpadky z pohľadu dát tečúcich havarovaným uzlom. Majme uzol U, ktorý má množinu obsahujúcu n vstupných prúdov dát (I_1, .., I_n) a produkuje výstupný prúd O. Výpočet e je postupnosť udalostí ako prijatie, spracovanie alebo vyprodukovanie n-tice dát. Výsledkom výpočtu e na uzle U je prúd dát O_e. Typy zotavenia [5] môžeme následne rozdeliť podľa spôsobu, akým sa snažia o dosiahnutie rovnosti O_f+O'=O, kde O_f je výsledok výpočtu pred haváriou, O' je výstup výpočtu po zotavení z výpadku a O prúd dát pri behu bez výpadku.

Gap recovery je najjednoduchším a zároveň najmenej náročným spôsobom zotavenia. Po detekcii výpadku uzlu zabezpečí jeho nahradenie, avšak negarantuje zachovanie vstupov a výstupov čo môže viesť k strate dát. Výhodou je rýchly čas zotavenia a minimálny vplyv na bezporuchový beh výpočtu.

Rollback recovery zaručuje, že výpadok nespôsobí stratu informácie. Podľa typu dotknutých operátorov delíme rollback recovery ďalej na:

- *opakujúci* pokiaľ sa pri obnove opakovane generujú n-tice, ktoré sú zhodné s pôvodne odoslanými.
- *konvergujúci* ak sú opakovane generované n-tice (z rovnakých dát), avšak tieto nie sú zhodné s pôvodnými, ale postupom času skonvergujú k n-ticiam behu bez výpadku.
- *divergujúci* pokiaľ sa n-tice generované po zotavení nikdy neskonvergujú k n-ticiam ktoré by boli generované počas behu bez výpadku. Tento prípad typicky nastáva u nedeterministických operátorov.

Precise recovery poskytuje najsilnejší spôsob zotavenia. Kompletne maskuje výpadok a zaručuje že výstup je rovnaký ako v prípade bezchybného behu.

2.1 Klasifikácia operátorov

Podľa vplyvu na sémantiku obnovenia rozlišujeme štyri typy operátorov: *obecný*, *deterministický*, *konvergujúci* a *opakovateľný*. Typ plánu spracovania určuje najobecnejší operátor, ktorý sa v ňom vyskytuje.

Operátor nazveme *deterministický* ak produkuje rovnaký výstup stále, keď sa začne výpočet v rovnakom stave a na rovnakej postupnosti vstupných dát na každom zo vstupov. Zdrojom nedeterminizmu môže byť napríklad závislosť na čase, poradí príchodu dát medzi jednotlivými vstupmi, alebo použití nedeterministického spracovania (napríklad použitie náhodných veličín).

Deterministický operátor je *konvergentný* pokiaľ je schopný po reštarte schopný konvergentného zotavenia, ak obnovu začína z prázdneho stavu a za pomoci opakovaného spracovania vstupných dát od určitého bodu v čase.

Konvergentný operátor je *opakovateľný* ak je schopný po reštarte schopný opakovateľného zotavenia, ak obnovu začína z prázdneho stavu a za pomoci opakovaného spracovania vstupných dát od určitého bodu v čase.

3 Metódy HA

Medzi základné prístupy dosiahnutia spoľahlivosti spracovania v distribuovaných systémoch patria *procesné páry (process-pairs)*, *logovanie* a *checkpointing*. V nasledujúcich podkapitolách predstavíme základné algoritmy založené na týchto postupoch. Aj keď v základných variantoch nedosahujú všetky uvedené postupy precíznej obnovy, je možné ich na takúto úroveň rozšíriť. Jedná sa hlavne o ošetrenie duplicitných stavov a zaručenie, že nedôjde k strate správ. Zabezpečenie proti strate riešia všetky metódy logovaním správ vo výstupných frontách dovtedy, kým nie je zaručené, že správa bola uložená alebo spracovaná. V prípade výpadku sa znovu posielajú takto zalogované správy. Ošetrenie duplicitných správ sa rôzni a preto je spomenuté podrobnejšie pri jednotlivých metódach.

3.1 Passive standby

Metóda passive standby [6, 5] je založená na *process-pairs* prístupe. Ku každému primárnemu uzlu je priradený záložný uzol. Primárny uzol v pravidelných intervaloch posiela aktualizáciu stavu na záložný. V prípade výpadku preberá záložný uzol výpočet a pokračuje od posledného checkpointu. Pre dosiahnutie precízneho obnovenia je nutné zabezpečiť, aby neboli vygenerované duplicitné dáta a opakovane spracovať správy prijaté havarovaným uzlom, ktoré nestihol reflektovať v zálohe. Prvý problém je riešený opýtaním sa nasledujúcich uzlov na správy, ktoré prijali a tieto sa už nebudú posielať. Druhý problém riešia výstupné fronty, ktoré ukladajú správy dovtedy, kým nie je potvrdená záloha stavu, ktorý ich obsahuje.

Výhodou tohto prístupu je krátky čas potrebný na obnovenie výpočtu, ktorý zodpovedá opakovanému spracovaniu správ prijatých od posledného checkpointu, avšak za cenu spomalenia riadneho behu bez výpadkov z dôvodu vyčlenenia (minimálne) polovice uzlov ako záložných a zvýšenej komunikácie primárnych ulov so záložnými. Ďalšie zdržanie výpočtu nastáva pri synchrónnom zálohovaní (primárny uzol neodosiela dáta dovtedy, kým nie sú uložene aj na záložnom uzle). Pri asynchrónnom zálohovaní naproti tomu hrozí strata dát a nekonzistentný stav ak nie je použitá doplňujúca metóda na prevenciu straty dát.

3.2 Active standby

Active standby je ďalším variantom *process-pairs* [6, 5, 7], ale na rozdiel od passive standby nečaká záložný uzol pasívne, ale dostáva rovnaké dáta ako primárny uzol a aktívne tieto dáta spracováva. Výstup záložného uzlu je namiesto odosielania logovaný. Pre dosiahnutie precízneho obnovenia musí záložný uzol zistiť identifikátory posledných správ, ktoré boli prijaté nasledujúcimi uzlami, aby nedošlo k duplicitnému poslaniu správ. Ak plán obsahuje nedeterministické operátory je navyše nutné doplniť logovanie rozhodnutí týchto operátorov a toto následne odosielať na záložný uzol za cenu extra komunikácie.

Výhodou active standby oproti passive variante je minimálny potrebný čas na obnovenie, pretože záložný uzol dostáva rovnaké správy, ktoré paralelne spracováva. To má avšak za následok nárast komunikácie o 100% . Ďalšiu extra komunikáciu znamená potvrdzovanie prijatých správ, slúžiacich pre orezanie výstupných zásobníkov a v prípade nedeterministických operátorov posielanie rozhodovacích záznamov.

3.3 Upstream backup

Klasické process pair prístupy majú veľký runtime overhead (je nutné vyčleniť aspoň polovicu uzlov ako záložné uzly) a vyžadujú nezanedbateľné navýšenie komunikácie. Upstream backup [6, 5] je naproti tomu navrhnutý tak, aby lepšie využil distribuovaný charakter prúdového spracovania dát. Upstream uzly (uzly umiestnené proti prúdu dát) slúžia ako zálohy pre downstream uzly (uzly ďalej po prúde dát), pričom sa zálohujú iba n-tice posielané týmto uzlom. V prípade výpadku uzlu, je tento nahradený novým uzlom s prázdnym stavom. Znovu spracovaním n-tic uložených na upstream uzloch dôjde k obnoveniu stavu uzlu.

Hlavným problémom tohto prístupu je narastajúca veľkosť výstupných front, ktoré slúžia ako zálohy. Pre ich zmenšenie je definovaný ich orezávací protokol (queue trimming protocol), ktorý popisuje kedy môžu byť n-tice z jednotlivých výstupných front odstránené. Cieľom je dosiahnuť minimálnu množinu, ktorá bude dostačujúca pre obnovenie stavu havarovaného uzlu. Orezávací protokol je

založený na potvrdzovaní doručenia, resp. spracovania n-tic. Informácie o prijatí sú šírené spätne pomocou potvrdzovacích správ, pričom existuje niekoľko úrovní týchto správ. Základná nultá úroveň oznamuje, že daná n-tica bola prijatá príjemcom. Uzol, ktorý prijme potvrdenie nultej úrovne, spätne vyráta ktoré vstupné n-tice sa podieľali na vytvorení potvrdenej n-tice a pošle potvrdenie prvej úrovne. Uzol ktorý prijme potvrdenie prvej úrovne pre konkrétnu n-ticu od všetkých uzlov, kam bola odoslaná, môže následne odstrániť túto a všetky logicky predchádzajúce n-tice. Pokiaľ sa požaduje vyšší stupeň zabezpečenia, iteratívne sa zvyšuje úroveň potvrdenia a orezávanie prebieha pri prijatí najvyššej požadovanej úrovne.

Výhodou tohto prístupu je zmenšenie nadmerného využívania prenosového pásma, keďže navyše používa iba potvrdzovacie správy, ktoré sú menšie než správy dátové a k extra prenosu dát dochádza len pri zasielaní n-tic po výpadku. Nevýhodou je neúmerne dlhšia doba potrebná na zotavenie, keďže nový uzol musí znovu spracovať dostatok n-tic pre obnovenie svojho vnútorného stavu a náročnosť výpočtu potvrdení vyšších úrovní.

3.4 Cooperative passive standby

Cooperative passive standby [8] je prístup založený na jemne členenom checkpointovaciom modeli. Ten je použitý, pretože efektívne funguje pre väčšiu skupinu úloh a prípadov než ostatné alternatívy ako znovu spracovanie, či redundantné spracovanie.

Plán(y) spracovania každého uzlu sú rozdelené do viacerých nezávislých HA jednotiek. Tieto sú následne rozdistribuované a vybalansované medzi viacerými servermi. Toto rozdelenie umožňuje rozložiť záťaž zálohy jedného uzlu medzi skupinu uzlov a týmto docieliť rýchlejšiu obnovu v prípade výpadku.

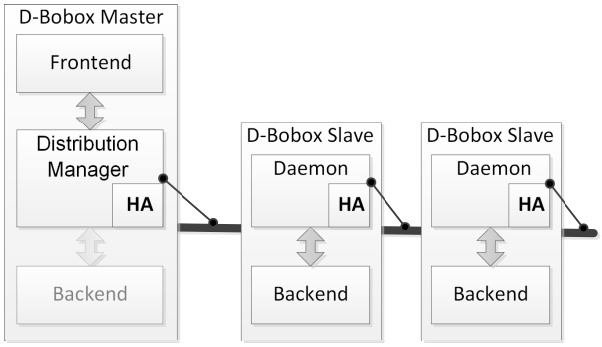
Zálohovanie prebieha po jednotlivých HA jednotkách na danom uzle, čo redukuje dĺžku blokovania výpočtu spôsobené zálohovaním HA jednotky. Jemnejšia granularita záloh umožňuje využiť voľné cykly CPU daného uzlu, keď práve neprebieha výpočet (napr. z dôvodu čakania na dáta) a týmto optimalizovať celkový výkon. Samotná záloha prebieha v dvoch krokoch: capture a paste. Počas fázy capture dôjde na uzle k výberu HA jednotky, ktorá sa bude zálohovať a odoslaniu správy na záložný uzol s aktualizáciou stavu danej HA jednotky. Paste fáza prebieha na záložnom uzle, ktorý spracúva doručené zálohovacie správy tak, že aktualizuje zodpovedajúci obraz checkpointu aplikovaním rozdielových informácii. Po ukončení fázy paste je upovedomený odosielateľ a môže dôjsť k ďalšiemu kolu zálohy tejto HA jednotky.

Po výpadku uzlu preberú jeho úlohu uzly obsahujúce zálohy jeho jednotlivých HA jednotiek. Počas obnovy dôjde k operácii paste a spracovaniu nespracovaných aktualizácii, pokiaľ také sú. Ďalej sú prepojeé vstupy z havarovaného uzla na záložné a spracované správy z výstupných front, ktoré boli odoslané havarovanému uzlu ale neobsahuje ich záloha. Rozdelenie úlohy havarovaného uzlu medzi viacero záložných znamená, že očakávaný nárast záťaže týchto záložných uzlov bude malý a oneskorenie výpočtu bude pod kontrolou.

Ak dôjde k obnoveniu havarovaného uzlu, tento je pripojený ako nový prázdny uzol a v rámci automatického vyvažovania záťaže môže byť na neho prevedená časť úloh. Vytváranie jednotlivých HA jednotiek a ich rozdelenie na záložne servery môže prebiehať automaticky, bez nutnosti pevnej konfigurácie, čo umožňuje dynamické prispôsobenie sa aktuálnej situácii a rozloženiu záťaže.

4 D-Bobox

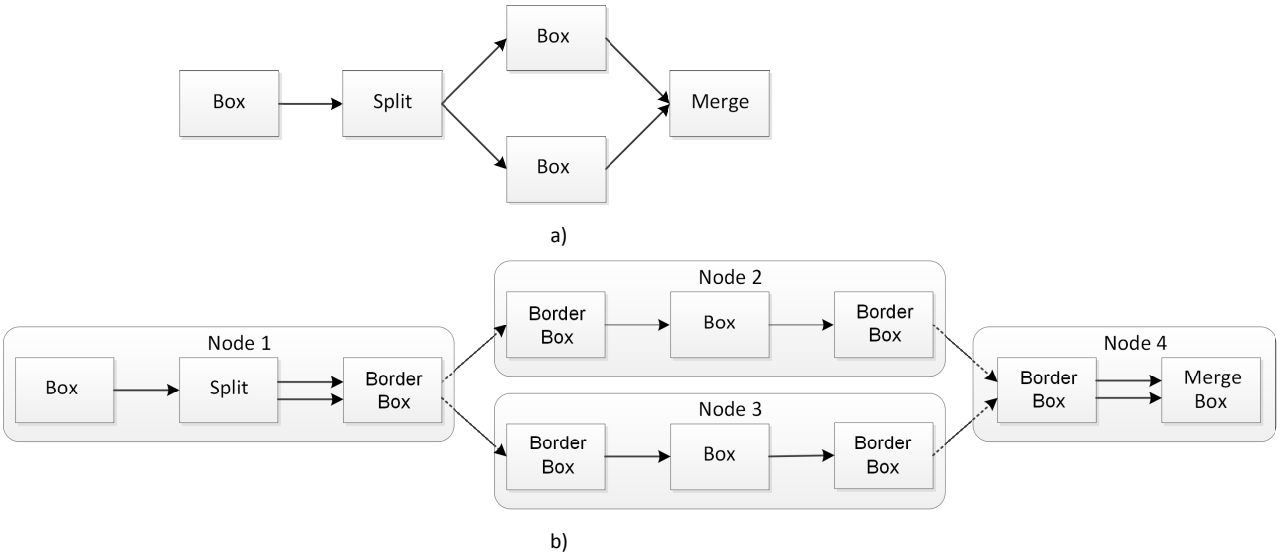
Bobox [9, 3] je framework pre vytváranie aplikácií určených k spracovaniu veľkých dát v paralelnom prostredí. Základný princíp spracovania je založený na rozdelení úlohy na menšie, vzájomne prepojené časti, ktoré môžu byť vykonávané nezávisle. Toto rozdelenie definuje plán spracovania a je reprezentovaný ako acyklický orientovaný graf, v ktorom uzly (v Boboxe nazývané krabičky) reprezentujú jednotlivé dielčie výpočty a orientované hrany reprezentujú tok dát. Dáta prúdia z jednotlivých zdrojových krabičiek do výstupnej krabičky. Zdroje dát môžu byť nielen statického charakteru, ako napríklad databáze, ale aj dynamické ako rôzne senzory, kamery atď. Táto univerzálnosť umožňuje nasadenia Boboxu na široké spektrum úloh.



Obr. 1: Architektúra systému D-Bobox.

D-Bobox [10] je ďalším evolučným krokom systému Bobox, keď k zvýšeniu efektivity spracovania veľkých dát už nepostačuje lokálny paralelizmus a je nutné expandovať do distribuovaného prostredia. Zároveň je snaha zachovať širokú použiteľnosť takto rozšíreného systému a preto je budovaný s ohľadom na efektívnu použiteľnosť aj na bežne dostupnom hardvéri a nielen na špecializovaných výpočtových clustroch.

Základná schéma D-Boboxu je znázornená na obrázku 1. Uzly sú rozdelené na jeden hlavný - master uzol, ktorý prijíma požiadavku na úlohu, vytvára plán jej spracovania (frontend časť), inicializuje podriadené - slave jednotky na spracovanie a (typicky) je aj cieľom výsledkov spracovania. Podriadené uzly obsahujú štandardný backend z Boboxu a sú rozšírené o potrebnú distribučnú lo-



Obr. 2: Ukážka rozšírenia exekučného plánu do distribuovaného prostredia pomocou hraničných krabičiek. a) Bobox plán pre výpočet na jednom uzle. b) Plán D-Boboxu, rozšírený o hraničné krabičky (Boundary Box), ktoré zaisťujú vzdialenú komunikáciu (prerušovane).

giku. Vzdialená komunikácia je riešená pridaním špecializovaných hraničných krabičiek, ktoré sú do výsledného plánu pridávané za behu, tesne pred inicializovaním pracovných uzlov. Pridáva a konfiguruje ich distribučná riadiaca jednotka na primárnom uzle podľa aktuálnej dostupnosti uzlov zúčastňujúcich sa výpočtu. Príklad takéhoto rozšírenia exekučného plánu ilustruje obrázok 2.

5 HA a D-Bobox

5.1 Základné algoritmy a D-Bobox

Každý z HA algoritmov spomenutých v predchádzajúcej kapitole je nasaditelný v systéme D-Bobox, avšak kladie rôzne požiadavky na samotný systém a poskytuje rôznu úroveň transparentnosti z pohľadu tvorby nových operátorov (krabičiek v systéme Bobox). V nasledujúcej diskusii neberieme do úvahy nedeterministické plány, ktoré pre všetky algoritmy vynucujú zo strany krabičiek podporu logovania nedeterministických rozhodnutí a tieto vhodným spôsobom reportovať, čo znižuje požadovanú transparentnosť.

Upstream backup predstavuje najmenej blokujúci prístup a vyžaduje najmenšiu extra komunikáciu počas behu bez výpadkov, avšak jeho nároky na ukladanie výstupných front môžu byť neúnosne veľké a cena obnovy uzlu v prípade výpadku je veľká z pohľadu času aj potrebného výpočtového výkonu. Navyše predstavuje najmenej transparentný prístup, pretože jednotlivé operátory musia implementovať často netriviálne mapovania medzi výstupnými a vstupnými n-ticami, ktoré je potrené pre korektné fungovanie potvrdzovania a orezávacieho algoritmu na výstupné fronty. Preto je nevhodný pre Bobox, ktorý sa snaží maximálne skryť paralelizačnú a distribučnú logiku z krabičiek za účelom ich jednoduchého vývoja.

Active standby je naproti tomu maximálne transparentný a takéto spracovanie môže byť dokonca zachytené upraveným plánom spracovania, kde budú jednotlivé časti spracovania zduplikované a výstupy paralelných vetiev budú smerované na špeciálnu systémovú krabičku, ktorá zabezpečí opätovné poslanie dát tak, aby po výpadku nedošlo k strate dát (ak výpočet v primárnej vetve zaostával za záložnou) a ani k produkcii duplicitných n-tic (ak výpočet v záložnej vetve zaostával za primárnou vetvou). Úlohou riadiacej časti HA je v prípade výpadku nájsť náhradný uzol za havarovaný, ktorý preberie úlohu záložného uzlu, zreplikovať na neho stav zo záložnej vetvy (po výpadku sa z nej stáva vetva primárna) a vhodne presmerovať komunikačné kanály. Tento HA algoritmus môže slúžiť pre úlohy preferujúce minimálny čas zotavenia pred výkonom, pretože je nutné obetovať polovicu uzlov ako uzly záložné. Rovnaká nevýhoda platí pre passive standby algoritmus, ktorý navyše nedosahuje podobne rýchle obnovenia, z dôvodu opätovného spracovania správ ktoré nezachytil posledný checkpoint a spomalenia výpočtu počas vykonávania zálohy. Z tohto dôvodu nie je táto možnosť uvažovaná pre Bobox.

Kooperatívny variant passive standby patrí takisto medzi transparentné metódy a jeho výhodou je rozumne malý vplyv na bezporuchový beh a rýchlosť zotavenia z výpadku. Preto poslúži ako základ pre integráciu HA v systéme D-Bobox.

5.2 Integrácia HA v D-Boboxe

Podpora HA je z pohľadu umiestnenia rozdelená na dva časti: lokálnu, umiestnenú na jednotlivých pracovných uzloch a globálnu, nachadzajúcu sa na primárnom uzle. Obe časti sú reprezentované zodpovedajúcim managerom a riešia odlišné úlohy: lokálny manager rieši správu HA v

kontexte jedného uzlu a globálny riadi HA na úrovni celého systému.

HA manager na primárnom uzle je tzv. globálny HA manager. Jeho zameraním je riešenie globálnych úloh ohľadne poskytnutia spoľahlivosti a efektívneho výpočtu: riešenie výpadkov, rozvrhovanie záloh a spolupráca s vyvažovaním záťaže.

V prípade výpadku je nutné informovať uzly držiace zálohy jednotlivých HA jednotiek havarovaného uzlu, aby vykonali paste operáciu doručených aktualizácií, ak tak ešte nevykonali a prebrali výpočet havarovaných jednotiek. Pre pokračovanie výpočtu je ešte nutné presmerovať posielanie správ na záložné uzly a informovať upstream uzly, aby odoslali uložené správy zo svojich výstupných front pre obnovenie stavu výpočtu, ktorý nestihol byť zachytený posledným checkpointom.

Vzhľadom na snahu o rozdelenie úlohy jedného uzlu na viac nezávislých HA jednotiek, ktoré sú distribuované na rôzne fyzické uzly, môžeme očakávať, že nárast záťaže týchto uzlov po tom ako preberú úlohu havarovaného uzlu bude v akceptovateľných medziach. Po obnovení havarovaného uzlu je tento pripojený k výpočtu a môžu byť na neho dynamicky preplánované úlohy a zálohy, typicky z uzlov ktoré sú najviac vyťažené (avšak je možné nasadiť rôzne stratégie zahŕňajúce napríklad lokalitu, cenu atď.). Hlavným dôvodom, prečo by sa HA manager mal snažiť o vyrovnávanie záťaže, je snaha dosiahnuť rozumne malé a časté zálohy všetkých uzlov. Pri nerovnomernej záťaži nevyťažené uzly produkujú zálohy častejšie čím pridávajú úlohy vyťaženým uzlom. Tieto naopak, aby nezdržovali výpočet redukujú interval svojich záloh a zálohy vykonávané po dlhšom čase bývajú obecne drahšie a obnovenia pomalšie, pretože došlo k väčším zmenám stavu, ktoré je treba pokryť. Snaha o dynamické vyrovnanie záťaže znižuje cenu checkpointov, keďže tieto sú menšie a tak sa s väčšou pravdepodobnosťou vojdú do voľných cyklov CPU a sú častejšie čo skracuje čas obnovy, keďže je nutné znovu spracovať menší počet n-tic pre obnovenie stavu. Vyrovnávanie záťaže docieľuje HA manager dvoma spôsobmi: vhodným rozvrhovaním zálohovacích párov a spoluprácou s riadiacou distribučnou jednotkou, ktorá riadi vyvažovanie pomocou migrácie úloh.

HA manager na sekundárnom uzle je lokálny manager. Má za úlohu riešiť HA úlohy spojené s konkrétnym uzlom medzi ktoré patrí: sledovanie dostupnosti susedov (upstream a downstream uzlov), správa HA jednotiek (delenie, zlučovanie) a plánovanie a vykonávanie operácií capture a paste.

Každý uzol sleduje dostupnosť susedných uzlov s ktorými komunikuje. Pokiaľ s nejakým susedným uzlom práve neprebieha komunikácia (netečú dáta), tak testuje jeho dostupnosť pomocou dotazov v pravidelných intervaloch. Ak nejaký uzol prestane reagovať, je po niekoľkých pokusoch (podľa nastavenej tolerancie) prehlásený za havarovaný a jeho výpadok je oznámený globálnemu HA managerovi, ktorý sa ujme riadenia nápravy tohto výpadku.

Počas behu bez výpadkov sú vykonávané operácie capture a paste. Capture má za úlohu zachytiť zmenu stavu konkrétnej HA jednotky a jej odoslanie na záložný server. Operácia paste ma za úlohu aktualizovať obrazy záloh pomocou doručených správ. Jednotlivé HA jednotky sú zálohované pomocou operácie capture nezávisle na sebe, zatiaľ čo počas operácie paste sa zálohy aktualizujú postupne podľa poradia prijatých správ dovtedy, kým sú nejaké nespracované aktualizačné správy, alebo pokiaľ nedôjde k preplánovaniu. Plánovanie operácií capture a paste má na starosti lokálny plánovač. Tento môže implementovať rôzne stratégie za účelom rozumne zálohovať, alebo minimalizovať blokovanie výpočtu.

Pre úpravu granularity checkpointingu umožnuje lokálny HA manager delenie, resp. zlučovanie HA jednotiek. Delenie je možné v prípade, ak uzol pracuje s viacerými krabičkami (nie nutne priamo prepojenými). Zjemnením granularity, hlavne u plánov obsahujúcich viacero netriviálnych operácií, dôjde k rozdeleniu záťaže zálohy a obnovenia na viacero záložných serverov. Naopak zlúčenie je vhodné pokiaľ sa na uzle ocitne niekoľko za sebou idúcich výpočtovo jednoduchých krabičiek, ktorých spojené vykonávanie nepredstavuje výrazný nárast záťaže (skôr dôjde k rýchlejšiemu spracovaniu vďaka lokalite vykonávania) a zmenši počet HA jednotiek, ktoré je potrebné plánovať. Tieto zmeny granularity sa musia vykonávať v spolupráci s globálnym HA managerom, ktorý musí novým HA jednotkám priradiť uzly, kam sa budú zálohovať a naopak zabezpečiť odstránenie záloh zrušených HA jednotiek.

HA na hraničných krabičkách musí pre garanciu precíznej obnovy podporovať logovanie výstupných správ a elimináciu duplicitných správ. Odchádzajúce správy sa logujú do doby, než je potvrdená ich záloha v HA jednotkách ktorým boli odoslané. Toto je nutné pre obnovenie stavu výpočtu po obnove z posledného checkpointu pomocou ich opakovaného spracovania. Vstupné hraničné krabičky musia taktiež pomocou identifikátorov správ filtrovať správy duplicitné, ktoré mohli pri opakovanom spracovaní vzniknúť (havarovaný uzol mohol a nemusel vyprodukovať pôvodné správy pred svojím pádom).

Integrovanie spomenutých častí poskytne základnú podporu vysokej dostupnosti v systéme D-Bobox. Táto môže byť následne rozšírená o podporu nedeterministických operátorov za pomoci logovania ich rozhodnutí, alebo napríklad vylepšovaná pridaním rôznych stratégií pri plánovaní záloh, či vyvažovaní záťaže.

6 Záver

V tomto článku sme prezentovali problém výpadku uzlu v distribuovaných systémoch prúdového spracovania dát,

definovali typy obnovenia a prezentovali súčasné základné prístupy k dosiahnutiu vysokej dostupnosti týchto systémov. Následne sme rozobrali vhodnosť jednotlivých prístupov pre systém D-Bobox a zvolili jeden z prístupov ako základný kameň, na ktorom bude zavedená podpora vysokej dostupnosti v D-Boboxe a načrtli možnosti jej integrácie do jednotlivých časti D-Boboxu.

7 Poď akovanie

Článok bol podporovaný Grantovou agentúrou Univerzity Karlovy, projekt č. 472313, grantom SVV-2013-267312, a projektom GAČR č. P103/13/08195S.

Literatúra

[1] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. S. Maskey, A. Rasin, E. Ryvkina *et al.*, "The design of the borealis stream processing engine." CIDR, 2005.

[2] D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, C. Erwin, E. Galvez, M. Hatoun, A. Maskey, A. Rasin *et al.*, "Aurora: a data stream management system," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. ACM, 2003, pp. 666–666.

[3] D. Bednarek, J. Dokulil, J. Yaghob, and F. Zavoral, "Bobox: Parallelization Framework for Data Processing," in *Advances in Information Technology and Applied Computing*, 2012.

[4] Z. Falt, J. Dokulil, M. Cermak, and F. Zavoral, "Parallel sparql query processing using bobox," *International Journal On Advances in Intelligent Systems*, vol. 5, no. 3, pp. 302–314, 2012.

[5] J.-H. Hwang, M. Balazinska, A. Rasin, U. Cetintemel, M. Stonebraker, and S. Zdonik, "High-availability algorithms for distributed stream processing," in *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, 2005, pp. 779–790.

[6] J. hyon Hwang, M. Balazinska, A. Rasin, U. Çetintemel, M. Stonebraker, and S. Zdonik, "A comparison of stream-oriented high-availability algorithms," Brown CS, Tech. Rep., 2003.

[7] M. A. Shah, J. M. Hellerstein, and E. Brewer, "Highly available, fault-tolerant, parallel dataflows," in *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '04. New York, NY, USA: ACM, 2004, pp. 827–838. [Online]. Available: http://doi.acm.org/10.1145/1007568.1007662

[8] J.-H. Hwang, Y. Xing, U. Cetintemel, and S. Zdonik, "A cooperative, self-configuring high-availability solution for stream processing," in *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*. IEEE, 2007, pp. 176–185.

[9] D. Bednárek, J. Dokulil, J. Yaghob, and F. Zavoral, "Data-flow awareness in parallel data processing," in *Intelligent Distributed Computing VI*, ser. Studies in Computational Intelligence, G. Fortino, C. Badica, M. Malgeri, and R. Unland, Eds. Springer Berlin Heidelberg, 2013, vol. 446, pp. 149–154. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-32524-3_19

[10] M. Cermak, Z. Falt, and F. Zavoral, "D-bobox: O distribuovatelnosti boboxu," in *Informačné Technológie - Aplikácie a Teória*. PONT s. r. o., 2012, pp. 41–46.