Fault tolerance systém v prostředí výpočetního Gridu

Fault Tolerance Sytem in Computional Grid

Petr Lukašík
Department of Computer and Communication Systems, Faculty of Applied Informatics
Tomas Bata University in Zlin
nam. T. G. Masaryka 5555, 760 01 Zlin, Czech Republic
plukasik@tajmac-zps.cz

Martin Sysel
Department of Computer and Communication Systems, Faculty of Applied Informatics
Tomas Bata University in Zlin
nam. T. G. Masaryka 5555, 760 01 Zlin, Czech Republic
sysel@fai.utb.cz

Abstrakt
Tolerance chyb je důležitá vlastnost, zejména v rozsáhlých výpočetních systémech, kde geograficky distribuované a vzdálené uzly spolupracují na provedení úkolu. Vysoká úroveň spolehlivosti a dostupnosti, je podmínkou odolnosti proti chybám. Selhání jakékoliv větve této struktury má zásadní vliv na QoS (Quality of Service) soustavy [7]. Hranice odolnosti systému proti poruchám má zásadní vliv na výkonové parametry systému. V článku je také popsána praktická realizace principu řešení chybových stavů v prostředí Gridu.

Abstract
Fault tolerance is an important feature, especially in the large-scale computing systems. This method defines rules for cooperation and the result management in the geographically remote nodes. Fault tolerance is the main mechanism for increasing reliability of the system. Failure of any part in the structure has a major impact on QoS (Quality of Service) [7]. Setting boundaries of the fault tolerance has a major impact to performance and availability of the system. The article also describes the practical realization of the principle of resolving error conditions in a Grid environment.

INTRODUCTION

Tolerance chyb je schopnost systému plnit svou funkci správně i za přítomnosti poruch. Odolnost proti chybám je základní vlastností, která vede k vyšší spolehlivosti. Primární metodikou pro zvýšení spolehlivosti je prevence. Kontrolní mechanismy, jejichž smyslem je odstranit okolnosti, kterými vznikají poruchy, jsou velmi důležité při provozu rozsáhlých infrastruktur [8].

Každý systém má definovány parametry, které zaručují bezchybnou funkcionalitu. Základním parametrem z pohledu QoS je definovaná šířka přenosového pásma. Jakákoliv odchylka od normálního chování má vliv na šířku přenosového pásma soustavy. K selhání tedy

dochází, když se skutečné chování systému odchyluje od definice standardních parametrů. Chyba představuje neplatný stav, který není v souladu se specifikací konkrétní aplikace. Jinými slovy, chyba je příčinou selhání systému. Standardně používanými technikami pro zvýšení odolnosti proti chybám jsou checkpointing a replikace. Obě tyto techniky však mají vliv na vyšší alokaci výpočetních zdrojů. Diskutovanými pojmy v oblasti fault tolerance jsou odolnost systému proti chybám a vysoká dostupnost systému. Rozdíl mezi oběma pojmy lze definovat následovně:

- Systém odolný proti chybám má velmi malou pravděpodobnost vzniku poruchy, ale výrazně vyšší režii provozu.
- Vysoce dostupný systém má vyšší pravděpodobnost vzniku poruchy a zároveň nižší režii provozu.

Požadavky na systém odolný proti chybám

- Porouchaná část nesmí způsobit výpadek systému (No single point of failure)
- Izolace chyb (fault containment) – chyba se nesmí dále v systému šířit
- Dostupnost režimů návratu – systém musí umět obnovit svoji činnost od jistých definovaných bodů návratu (checkpoint)
- Pokud má systém poruchu, musí pokračovat v činnosti i během rollback procesu.

AKTIVNÍ A PASIVNÍ REPLIKACE

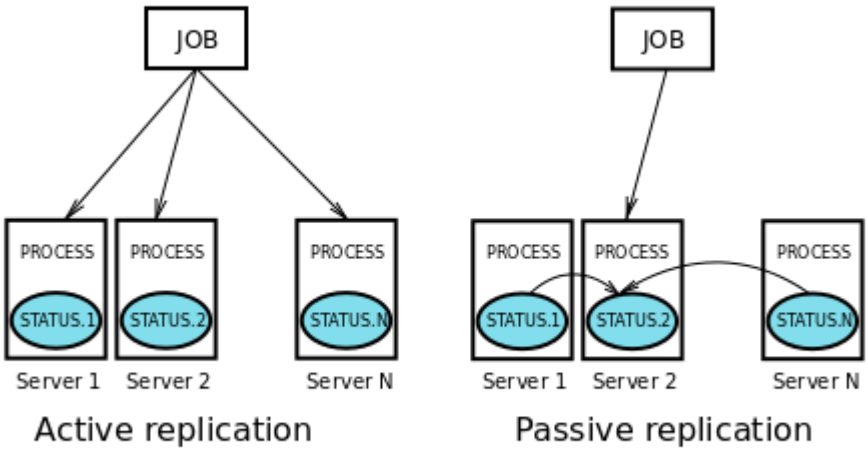
Replikace je technika založená na předpokladu, že každý jednotlivý výpočetní zdroj je mnohem náchylnější k poruchám, než současné selhání více zdrojů naráz. Principem replikace je spuštění několika kopií stejného úkolu na více než jednom výpočetním zdroji. Počet replikací je přímo úměrný požadavku nárůstu spolehlivosti a nepřímo úměrný požadavku na výkonnost výpočetního systému. To znamená, že čím více je definováno replik jedné a téže úlohy, tím více narůstá odolnost systému vůči poruchám, ale tím také na druhou stranu klesá výkonnost systému a to nejen násobně vyšší potřebou výpočetních uzlů, ale také nárůstem režie systému a zatížení síťového provozu. Stanovení optimálního počtu replik představuje relativně obtížnou disciplínu a je předmětem mnoha technických a teoretických úvah [2]. Nárůst replik samozřejmě nezvyšuje odolnost systému lineárně. Nevhodná nebo zbytečně předimenzovaná volba počtu replikovaných uzlů vede k vyšší režii na správu a provoz celého systému.

Aktivní replikace je definována tak, že každá úloha je zadána do zpracování více výpočetním uzlům. Aktivní replikace (state machine replication) vyžaduje, aby procesy běžící ve výpočetním uzlu byly deterministické. To znamená, že s ohledem na stejný počáteční stav, budou všechny procesy produkovat stejnou sekvenci odezvy a budou končit ve stejném koncovém stavu. Pro aktivní replikaci je důležité, aby všechny operace byly atomické. To znamená, že se provedou správně a oznámí správný výsledek, nebo se neprovedou vůbec. Nevýhodou pro aktivní replikaci je, že reálný systém je nedeterministický. Aktivní replikace je vhodná pro real-time aplikace, kde deterministické chování systému je podmínkou. Typickým příkladem pro aktivní replikaci jsou disková pole, kdy dochází k online replikaci dat. Podmínkou pro tuto replikaci jsou disková pole stejných vlastností [10].

V případě pasivní replikace je definován pouze jeden server (primární), který zpracovává požadavky klientů. Po zpracování žádosti, primární server aktualizuje stavy na straně záložních serverů. Výsledek operace pošle zpět zadavateli úlohy. V případě, že primární server selže, sekundární server automaticky nastoupí na jeho místo. Výhodou pasivní replikace je, že může být

využita i pro nedeterministické procesy. Nevýhodou pasivní replikace je ve srovnání s aktivní replikací, delší odezva systému na vzniklou poruchu.

Vícenásobné zpracování jediné úlohy má také vyšší nároky na systém i celý síťový provoz. V případě, že v průběhu žádná chyba nevznikne, jsou nadbytečné větve výpočtu zahozeny. Výhodou proti checkpointingu je snadnější mechanismus obnovy vzniklé chyby. Představitelem tohoto typu replikace je například Hadoop cluster [11]. Princip aktivní a pasivní replikace je znázorněna na obrázku 1.



Obrázek 1: Principy replikace v distribuovaných systémech

JOB CHECKPOINTING

Principem checkpointingu je snímkování aktuálního stavu procesu a jeho zaznamenávání do nezávislého úložiště - žurnálu procesů. V případě, že vznikne jakákoliv porucha, systém dokáže provést rollback, to znamená restaurovat stav, který byl před poruchou. Následně tuto defektní část znovu pošle do zpracování. Rekonstrukce stavu distribuovaného systému s více procesy je poměrně velmi náročná úloha. Zejména u procesů, které vzájemně komunikují prostřednictvím zpráv, může rollback způsobit komplikace v integritě úlohy. Výhodou checkpointingu je (na rozdíl od replikace) nižší režie systému. Nevýhodou je vyšší složitost řešení mechanismů obnovy z poruchového stavu [2].

Rollback a recovery jsou techniky určené pro eliminaci transientních chyb systému. Transientní chyba může vzniknout kdykoliv v průběhu výpočtu. Tato chyba nemusí být detekována okamžitě. Příčinou vzniku takovéto chyby může být řada výjimek, jako například nestabilní síťové spojení, hardwarová nebo softwarová chyba. Transientní chyba je definována jako chyba dočasná a opravitelná (recoverable). Vznik transientní chyby znamená, že systém musí obnovit svůj stav do okamžiku posledního checkpointu a musí provést restart [12].

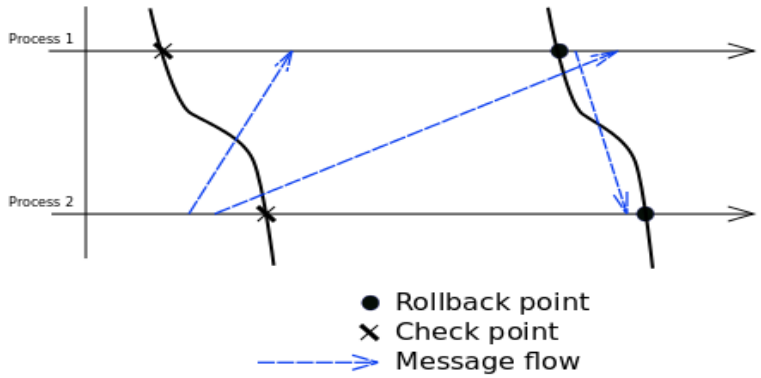
Checkpointing neřeší vznik permanentní chyby, která je definována jako neopravitelná (irrecoverable) a ve většině případů znamená výpadek celého systému.

V distribuovaných systémech, je systém posílání zpráv jedinou možností, jak zajistit komunikaci mezi jednotlivými procesy. Zpráva, která je generovaná odesílatelem, vyvolá determinované akce na straně příjemce. Úkolem checkpointingu je zajistit synchronizaci bodů obnovy (check pointů) a synchronizaci systému předávání zpráv. Tyto činnosti následně zajistí konzistenci prováděných transakcí[4].

Konzistentní operace checkpointingu.

Na obrázku 2. jsou znázorněny dva $\mathbf{P_i}$ a $\mathbf{P_j}$ procesy, které jsou na sobě závislé. To znamená, že chybný výsledek procesu **Pi** znamená automaticky také chybný výsledek procesu $\mathbf{P_j}$. Checkpointy $\mathbf{C_i}$ a $\mathbf{C_j}$ znázorňují recovery line (hranici obnovy), ze které budou procesy znovu obnovovat svoji činnost v případě poruchy. Rollback pointy $\mathbf{U_i}$ a $\mathbf{U_j}$ znázorňují rollback line (hranici rollbacku), z nichž procesy zahájí rollback. Úkolem rollbacku je obnovit stav do poslední recovery line, která je zde znázorněna body $\mathbf{C_i}$ a $\mathbf{C_j}$.

Jestliže $\mathbf{P_i}$ proces pošle zprávu procesu $\mathbf{P_j}$ o rollbacku musí i proces $\mathbf{P_j}$ provést rollback operaci. Rollback operace je spuštěna v okamžiku příjmu zprávy. Obecně nemůžeme předpokládat, že po restartu budou obnoveny i odchozí zprávy, které vznikly později, než byla definována recovery line, protože vyslání odchozí zprávy mohlo být způsobeno transientní chybou. Po restartu procesu jsou obnoveny všechny příchozí zprávy kromě zpráv, které vysílají procesy, na nichž se provádí rollback. [9]

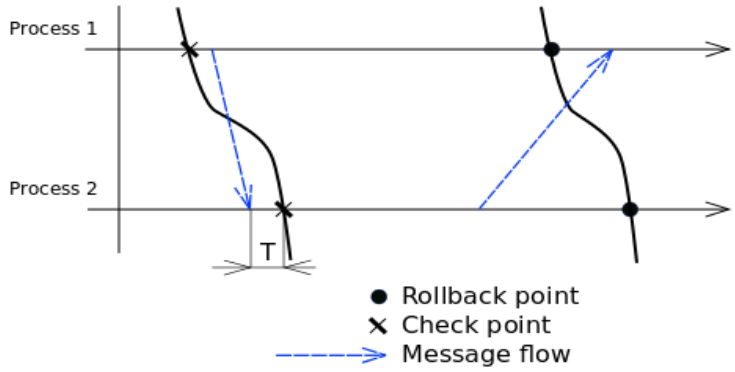


Obrázek 2: Konzistentní operace checkpointingu

Nekonzistentní operace checkpointingu.

Nekonzistentní stav nastane v okamžiku, kdy odesilatel vyšle zprávu o provedení rollbacku, ale příjemce tento rollback neprovede. Vznikne tak stav, který se nazývá visící příjem - dangling receive. Obrázek 2. znázorňuje nekonzistentní recovery line a nekonzistentní rollback line. Procesy $\mathbf{P_i}$ a $\mathbf{P_j}$ provádějí rollback do bodu $\mathbf{C_i}$ a $\mathbf{C_j}$. Proces $\mathbf{P_i}$ posílá zprávu **M** o rollbacku. Předpokládá se, že proces $\mathbf{P_j}$ provede všechny požadované akce, které jsou požadovány zprávou **M**. Ovšem zpráva m byla doručena procesu $\mathbf{P_j}$ dříve, než nastal bod $\mathbf{C_j}$. Proces se při rollbacku vrátí do bodu $\mathbf{C_j}$. To znamená, že část procesu v časové periodě **T** nebude obnovena. Proto recovery line $\mathbf{C_i}$,$\mathbf{C_j}$ je nekonzistentní. Podobně Rollback point $\mathbf{U_i}$,$\mathbf{U_j}$ je nekonzistentní, protože $\mathbf{P_j}$

vyšle zprávu l o rollbacku a P_i začne provádět rollback dříve, než je zpráva přijata [9]. Situace je znázorněna na obrázku 3.



Obrázek 3: Nekonzistentní operace checkpointingu

Konzistentní vlastnosti recovery line lze definovat následovně:

Definice 1:
Jestliže je zpráva přijata před checkpointem, musí být také odeslána před checkpointem. Jinými slovy: Sledovaný úsek procesu je stabilní, jestliže všechny zprávy vyslané tímto procesem před tímto sledovaným úsekem procesu, jsou stabilní.

Konzistentní vlastnosti rollback line jsou definovány takto:

Definice 2:
Jestliže některý z procesů vyšle zprávu o rollbacku, pak příjemci musí také vrátit zpět všechny akce, které byly vykonány do přijetí této zprávy.

Příklad distribuovaného procesu viz obr 4.

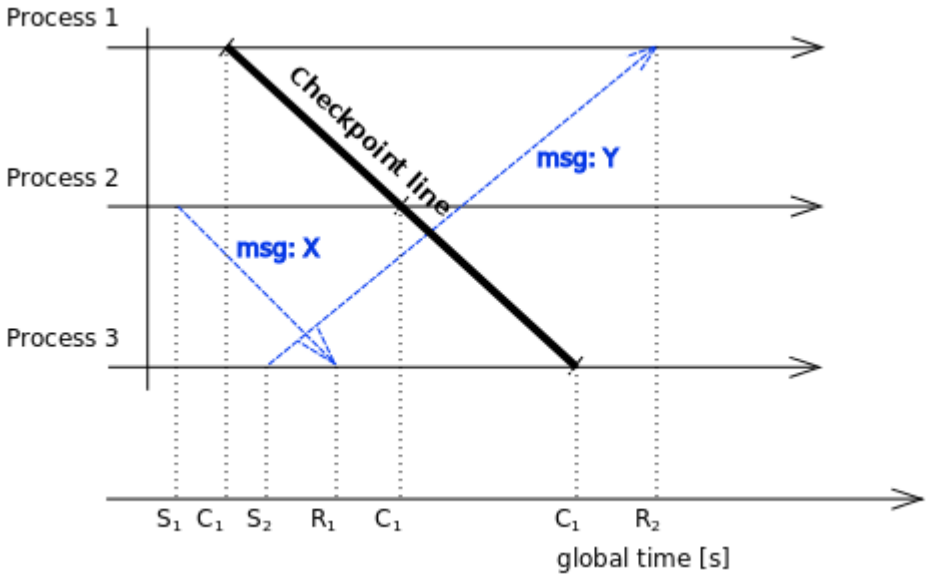
Procesy $\mathbf{P_i}$, $\mathbf{P_j}$ a $\mathbf{P_k}$ jsou znázorněny na časové ose. Body **C** představují checkpoint operace. Body **S** jsou body vyslání zprávy. **R** jsou body, kdy vyslané zprávy byly přijaty. Sekvence časového snímku všech operací jsou zaznamenány v logu:

$$L = S_2(j)C_1(i)S_3(k)R_2(k)C_1(j)C_1(k)R_3(i)$$

Subskripty reprezentují indexy transakcí. Hodnoty v závorkách jsou indexy procesů, které provádějí požadované operace. Tento log znázorňuje souběžný průběh tří transakcí

$$T_1 = C_1(i)C_1(j)C_1(k)$$
$$T_2 = S_2(j)R_2(k)$$
$$T_3 = S_3(k)R_3(i)$$

Transakce $\mathbf{T_1}$ (checkpoint transakce) je provedena nad všemi procesy systému. Tyto checkpoint operace definují checkpoint line. Transakce $\mathbf{T_2}$ (msg: X) posílá zprávu od procesu $\mathbf{P_2}$ do procesu $\mathbf{P_3}$. Transakce $\mathbf{T_3}$ (msg: Y) posílá zprávu z procesu $\mathbf{P_3}$ do procesu $\mathbf{P_1}$.



Obrázek 4: Příklad distribuovaného procesu

V případě vzniku chyby je úkolem systému obnovit stav – provést rollback do posledního konzistentního bodu systému. Konzistentní body systému jsou definovány na jednotlivých checkpointech [6], [9].

PRINCIPY CHECKPOINTINGU

Checkpointing na úrovni programu: Využívá se u aplikací, u nichž je k dispozici zdrojový kód. Princip zachytávání chybových stavů (try, catch) pak lze využít pro rozesílání zpráv a signálů. Výhodou je, že vlastnosti aplikace jsou plně pod kontrolou svých tvůrců. Nevýhodou je nutnost vynaložit programátorské úsilí [1].

Checkpointing na úrovni knihovny – DMTCP): je nástroj pro transparentní checkpointing souběžných aplikací, včetně aplikací distribuovaných aplikací. Tato knihovna spolupracuje přímo s binárními spustitelnými soubory. Nevyužívá ke své činnosti žádný z modulů linuxového jádra. DMTCP podporuje řadu aplikací na bázi: Matlab, Java, Python, Perl, Ruby, PHP, Ocaml, GCL (GNU Common Lisp), emacs, vi/cscope, Open MPI, MPICH-2, OpenMP [3].

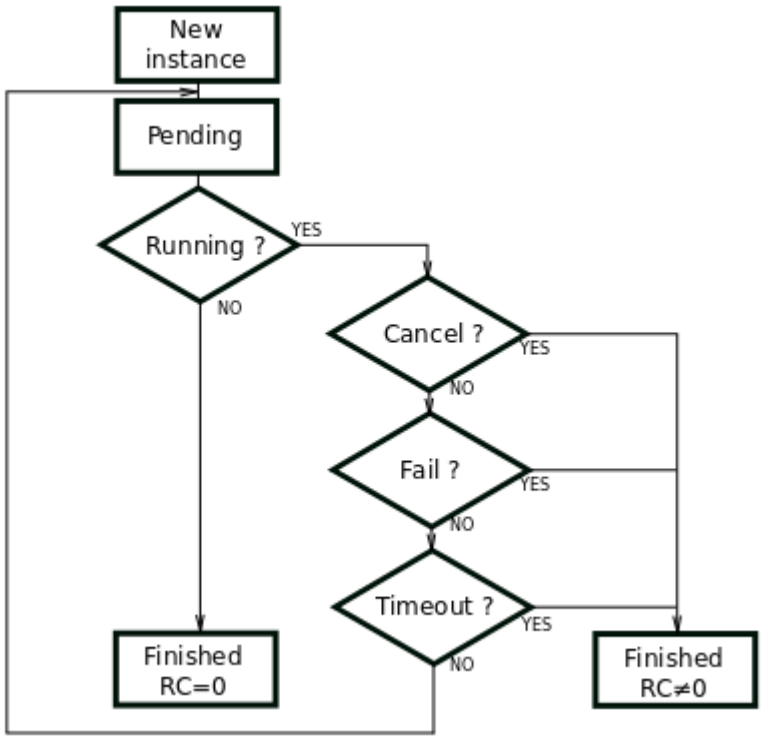
Checkpointing na úrovni kernelu systému (BLCR) - the Berkely Labs Checkpoint/Restart for Linux. Knihovna využívající systémové signály a zprávy (SIG...) linuxového kernelu. Nevýhodou je, že ji nelze portovat na jiný systém [13].

VYUŽITÍ FAULT TOLERANCE SYSTÉMU V APLIKACI INTRANET GRIDU

V prostředí intranet gridu se využívá princip checkpointingu na úrovni programu. Pro řízení úlohy jsou využity dvě komponenty JSDL (Job Submission Definition Language) a POSIX [14, 15].

JSDL je obecná specifikace pro řízení a distribuci dávkových úloh v prostředí Gridu. Současná verze 1.0 (vydána 07.11.2005) má také definovánu podporu POSIX. Úkolem fault tolerance je distribuce zpráv o stavu úlohy a jejich return kódech.

Životní cyklus úlohy představuje aktuální stav instance programu, který je právě vykonáván (nebo doby, než bude vykonán). Každá takto naplánovaná instance má přímý vliv na celkový výsledek. Informace o stavu úkolů, které jsou distribuovány jednotlivým klientům Gridu jsou velmi důležité pro plánovač, který je zodpovědný za úspěšné vyřešení úlohy. Přitom platí, že v řadě poruchových stavů nemusí být tato informace plánovači doručena. Stavy úlohy jsou popsány v obrázku 5.

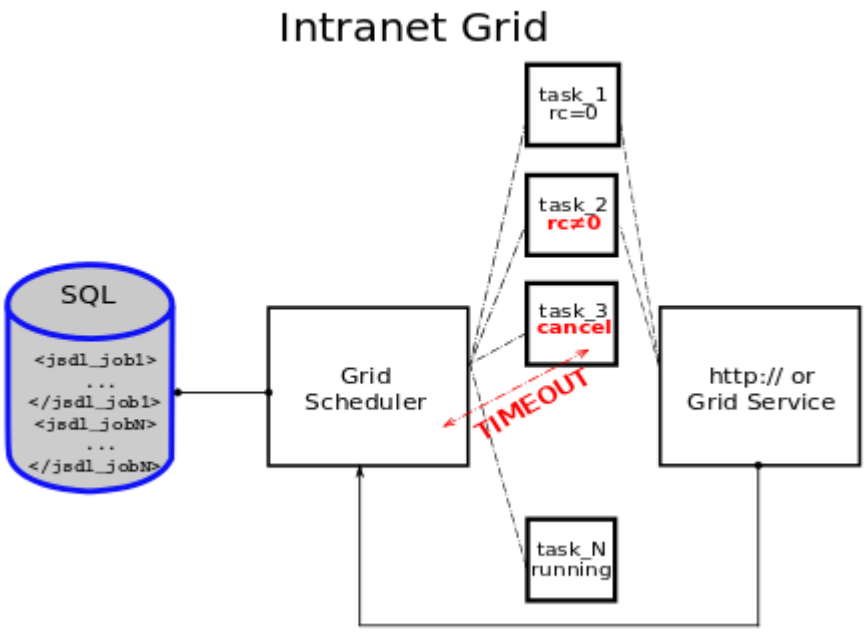


Obrázek 5: Diagram životního cyklu úlohy

Stejným životním cyklem lze také popsat celou úlohu, která je právě zpracovávána. Plánovač gridu musí mít přehled o každém stavu právě zpracovávané instance. Z diagramu vyplývá, že výsledek zpracování nabývá tři možné stavy. Tyto stavy lze ještě rozdělit do dvou skupin:

- Stav, kdy úloha dokáže vrátit zadavateli výsledek zpracování (správný nebo chybný)
- Stav, kdy úloha výsledek zpracování vrátit nedokáže, například pro přerušení běhu úlohy.

Stav kdy úloha nedokáže vrátit výsledek zpracování, musí být ošetřen na straně plánovače gridu. Princip je následující. Plánovač dostane informaci o přerušení komunikačního kanálu, nebo pomocí uživatelem zvolené maximální délky zpracování úlohy (timeout), kdy po uplynutí této doby je instance úlohy prohlášena za ztracenou. To pro plánovač znamená, aby tuto instanci poslal ke zpracování jinému klientovi. Existuje samozřejmě nebezpečí, že zdrojem nesnází na straně klienta je právě zpracovávaná úloha, která nemá správně řešeny chybové stavy, (memory leak, divide zero), pak se samozřejmě může stát, že takováto úloha může zapříčinit pád celé gridové infrastruktury. Proto i tento stav musí plánovač gridu řešit, například tak, že může tuto úlohu poslat do zpracování opakovaně jen několikrát, dle předem nastaveného scénáře [5].



Obrázek 6: Distribuce systémových zpráv v intranet gridu

JSDL má pro řízení úloh implementovánu podporu, která umožní definovat rozsahy výpočetních zdrojů (délka zpracování, počet spuštěných threadů, velikost diskového prostoru), které při překročení některého z parametrů úlohu nespustí. Dále lze definovat způsob komunikace s plánovačem Gridu při zpracování návratových kódů, případně výpadku některé ze spuštěných instancí, viz obrázek 6.

Relativně slabým článkem při komunikaci mezi plánovačem gridu a jeho klientskými uzly lze označit případ, kdy právě probíhající úloha je násilně ukončena. Stavy, které mohou způsobit pád, nebo násilné ukončení úlohy, nemusí v řadě případů vrátit chybovou informaci.

Definice fault tolerance systému a jeho vlastností má zásadní vliv na spolehlivost celé gridové služby. Úkolem je nejen zvýšit odolnost systému, ale i rozpoznat různé chybové scénáře a na základě nich zvolit vhodnou strategii obnovy. JSDL specifikace ve své definici pamatuje na některé kolizní a chybové stavy a umožňuje úloze předem nastavit některé limitní hodnoty, například velikost paměti, velikost souboru, počet vláken nebo maximální dobu běhu úlohy. To do

jisté míry umožňuje plánovači gridu, případně spuštěné úloze, lépe vyhodnotit mezní nebo chybové stavy a tím například vyvolat rollback úlohy případně ji znovu poslat do zpracování.

ZÁVĚR

Předložený text si klade za cíl popis principů, které umožňují zvýšit spolehlivost a odolnost proti poruchám v rozsáhlých distribuovaných systémech. Jako praktická ukázka bylo popsáno využití specifikací JSDL jako generátoru dávkových úloh a POSIXu, jako nástroje pro komunikaci mezi jednotlivými částmi zpracovávané úlohy.

JSDL parser pro intranet grid byl navržen netradičně tak, že přímo generuje skripty bash shellu, které jsou následně plánovačem a serverem gridu distribuovány do prostředí výpočetního gridu. Obecně je JSDL nasazováno v oblasti gridových služeb, které mají integrovány všechny funkcionality pro manipulaci s daty a komunikaci s okolím. Primární snahou bylo co nejvíce využít standardní a existující programové vybavení a systémové služby. To do jisté míry uživateli umožňuje vyšší kreativitu při návrhu a zpracování úlohy. V prvé fázi se také může zdát, že použití JSDL je v této oblasti zbytečně složité. Kód JSDL proti kódů výsledného shellového skriptu je na první pohled mnohem složitější a může se zdát, že je neefektivní. Výhoda JSDL je však okamžitě patrná v okamžiku, kdy potřebujeme například distribuovat tisíce paralelních instancí jediné úlohy.

Dalším směrem vývoje bude návrh a využití gridových služeb a JSDL pro úlohy v cloudovém clusteru Apache Hadoop. Ten se perspektivně jeví jako výborné datové úložiště velmi rozsáhlých datových struktur, kde problém násilně ukončené úlohy je řešen redundandním zpracováním a technologií Map Reduce.

REFERENCE

1. Lawall, J. L. & Muller, G., Efficient Incremental Checkpointing of Java Programs, INRIA Theme 2 Genie logiciel et calcul symbolique Projet COMPOSE Rapport de recherche n3810 Novembre 1999 23 pages, 1999
2. Altameem, T. Fault Tolerance Techniques in Grid Computing Systems T. Altameem / (IJCSIT) International Journal of Computer Science and Information Technologies, VOL. 4 (6) , 2013, 858 - 862, 2013
3. Cooperman, G. Checkpointing using DMTCP, Condor, Matlab and FreD; High Performance Computing Laboratory College of Computer and Information Science Northeastern University, Boston
4. Kumar, P.; Setiya, R. & Gahlan, P. Checkpointing Algorithms for Distributed Systems TECHNIA International Journal of Computing Science and Communication Technologies, VOL. 2, NO. 1, July 2009. (ISSN 0974-3375), 2009
5. Nandagopal, M. & Uthariaraj, V. R. Fault tolerant scheduling strategy for computational grid environment Malarvizhi Nandagopal et. al. / International Journal of Engineering Science and Technology VOL. 2(9), 2010, 4361-4372, 2010
6. Ropars, T. & Morin, C. An Extremely Optimistic Message Logging Protocol Inria - Rapport de recherche n 6357 — Novembre 2007, 2007
7. Enterprise QoS Solution Reference Network Design Guide Version 3.3, Cisco Systems Attn: Customer Document Ordering 170 West Tasman Drive San Jose, CA 95134-9883
8. Garg, R. & Singh, A. K. Fault Tolerance in Grid Computing: State of the art and open issues. International Journal of Computer Science & Engineering Survey (IJCSES) VOL.2, No.1, Feb 2011, 2011

9. Bharat, B. & Shy-Renn, L. Independent Checkpointing and Concurrent Rollback for Recovery in Distributed System - An Optimistic Approach ,Purdue University, Purdue e-Pubs Computer Science Technical Reports, Department of Computer Science, 1987
10. Lampert, L. Using Time Instead of Timeout for Fault-Tolerant Distributed Systems ACM Transactions on Programming Languages and Systems, VOL. 6, No. 2 April 1984, 254-280., 1984
11. White, T. Hadoop: The Definitive Guide O'REILLY (R) ISBN: 978-1-449-31152-0 1327616795
12. Pei-Jyun Leu & Bharat Bhargava Concurrent Checkpointing and Recovery in Distributed Systems, Purdue University Purdue e-Pubs Computer Science Technical Reports, 1987
13. Berkeley Lab Checkpoint/Restart (BLCR) User's Guide
14. Ali, A.; Fred, B.; Michel, D.; Donal, F.; Stephen, M.; Darren, P. & Andreas, S. Job Submission Description Language (JSDL) Specification, Open Grid Forum (2003-2005 2007-2008) © Open Grid Forum All Rights Reserved.
15. Group, POSIX.1-2008 The Open Group Base Specifications Issue 7 IEEE Std 1003.1™, 2013 Edition,The IEEE and The Open Group, 2013