УДК 004.021

МУЛЬТИАГЕНТНЫЙ АЛГОРИТМ ПЕРЕРАСПРЕДЕЛЕНИЯ ВЫЧИСЛИТЕЛЬНЫХ РЕСУРСОВ ДЛЯ ОСТАТОЧНОЙ СХЕМЫ РЕШЕНИЯ ЗАДАЧИ В GRID

Феоктистов А.Г., Костромин Р.О.

ФБГУН «Институт динамики систем и теории управления им. В.М. Матросова» СО РАН, Иркутск, e-mail: agf65@yandex.ru

В настоящее время обеспечение отказоустойчивости вычислительного процесса в распределенных вычислительных средах, например в Grid-системах или облачных инфраструктурах, по-прежнему остается актуальной проблемой. Целью нашего исследования является повышение отказоустойчивости процессов решения больших фундаментальных и прикладных задач в распределенных системах модульного программирования. Вычислительный процесс представлен абстрактной программой (схемой решения задачи). В статье предложен новый мультиагентный алгоритм для перераспределения вычислительных ресурсов Grid в случае отказа вычислительного процесса. Остаточная схема решения задачи формируется с использованием методов конкретизирующего программирования. В отличие от известных алгоритмов подобного назначения, предложенный алгоритм реализует адаптивное мультисценарное решение данной задачи и тем самым повышает степень отказоустойчивости вычислительного процесса. Работа алгоритма базируется на использовании конечно-автоматной модели эффективного и надежного взаимодействия агентов.

Ключевые слова: мультиагентная система, конечно-автоматная модель, управление вычислениями, смешанные вычисления, Grid

MULTI-AGENT ALGORITHM FOR REALLOCATION OF COMPUTATIONAL RESOURCES FOR RESIDUAL PROBLEM SOLVING SCHEME IN GRID

Feoktistov A.G., Kostromin R.O.

Matrosov Institute for System Dynamics and Control Theory of Siberian Branch of Russian Academy of Sciences, Irkutsk, e-mail: agf65@yandex.ru

Nowadays, a provision of computational process fault-tolerance in distributed computing environments such as Grid-systems or Cloud-infrastructures remains a relevant issue. The aim of our study is to increase the fault-tolerance for processes of solving large fundamental and applied problems in distributed systems of modular programming. The computational process is represented by an abstract program (a problem solving scheme). This paper proposes a new multi-agent algorithm for the Grid-resources reallocation, when the computational process is failure. The residual problem solving scheme is formed using methods of the abstract program specialization. In comparison to the known algorithms for the same purpose, the proposed algorithm implements an adaptive multi-scenario solving this problem, and therefore increases a degree of computational process fault-tolerance. The algorithm is based on using the finite state machine model of effective and reliable agents interaction.

Keywords: multi-agent system, finite state machine model, computation management, mixed computation, Grid

Проблема восстановления вычислительного процесса (задания) вследствие отказа программно-аппаратного обеспечения актуальна при решении больших фундаментальных и прикладных задач в разнородных распределенных вычислительных средах, в том числе в Grid различного назначения. Существуют различные подходы к решению данной проблемы [8].

В частности, широко применяемым на практике подходом к перезапуску заданий является механизм контрольных точек [7]. Однако формирование пользовательских контрольных точек в узле Grid, в котором выполняется вычислительный процесс, не позволяет произвести рестарт в случае отказа самого узла. Создание же системных контрольных точек на уровне операционной системы, обеспечивающих перенос задания в другие узлы, влечет существенные накладные расходы и поддерживается далеко не всеми системами управления Grid.

В статье рассматривается алгоритм восстановления вычислительного процесса после его отказа в Grid, где управление вычислениями реализуется мультиагентной системой [3]. В отличие от других подходов к управлению вычислениями и обеспечению их надежности [9, 10], наш подход базируется на реализации адаптивного мульти-сценарного алгоритма решения данной задачи, интегрирующего ряд эффективных и надежных алгоритмов взаимодействия агентов.

Иерархическая структура мультиагентной системы может включать два или более уровней функционирования агентов. На каждом уровне могут функционировать агенты, играющие различные роли и соответственно выполняющие различные

функции. Роли агентов могут быть постоянными и временными, возникающими в дискретные моменты времени в связи с необходимостью организации коллективного взаимодействия. Уровни иерархии агентов отличаются объемом знаний агентов – агенты более высокого уровня иерархии обладают большим объемом знаний по сравнению с агентами более низкого уровня иерархии и, кроме того, могут обращаться к агентам любого ниже лежащего уровня с запросом на получение локальных знаний этих агентов. На каждом уровне иерархии агенты могут объединяться в виртуальные сообщества, кооперироваться и конкурировать в рамках этих сообществ.

Вычислительная модель

Пусть Z, F и M – это множество параметров, операций и программных модулей вычислительной модели предметной области. Операции из F определяют отношения вычислимости на множестве параметров Z. Каждая операция $f_i \in F$ реализуется модулем $m_j \in M$, где $i \in \overline{1,n_f}$, $j \in \overline{1,n_m}$, n_f – число операций; n_m – число модулей. Один модуль может реализовывать несколько операций. С каждой операцией f_i связано два множества параметров $Z_i^{in}, Z_i^{out} \subset Z$. Множество Z_i^{in} определяет параметры, значения которых необходимо задать, чтобы получить значения параметров, представленных множеством Z_i^{out}. Множества Z_i^{in} и Z_i^{out} представляют соответственно множества входных и выходных параметров модуля m_j, реализующего операцию f_i.

Постановки задач могут формулироваться в полной или сокращенной (процедурной или непроцедурной) форме. По сформулированной постановке задачи строится схема решения задачи на основе сочетания волновых и переборных методов планирования вычислений [4]. В общем случае в вычислительной модели предметной области может существовать множество S эквивалентных схем решения задачи. Схема $s \in S$ определяет, какие операции и в какой последовательности должны быть выполнены для решения задачи.

Полная постановка задачи s_f, совпадающая со схемой s, определяется структурой $s_f = \langle F_s, X_0, Y_0 \rangle$, где $F_s \subset F$ – множество операций, которые нужно выполнить для решения задачи; $X_0 \subset Z$ – множество исходных параметров, значения которых заданы; $Y_0 \subset Z$ – множество целевых параметров, значения которых нужно вычислить.

Остаточная схема решения задачи

Пусть $c(P, X, Y)$ – вычислитель, выполняющий программы из множества P, где X – множество параметров программ из P, значения которых вычислены; Y – множество параметров программ из P, значения которых нужно вычислить. Тогда, согласно А.П. Ершову [6], программа $p_x(Y) \in P$ называется остаточной программой (проекцией p на x) такой, что $p_x(y) = p(x, y)$, $x \subset X$, $y \subset Y$.

Пусть $s \in P$ и вычислителем выполнено множество $F_r \subset F_s$ операций. Тогда, остаточной схемой $s_x(y) \in P$ решения задачи называется проекция s на x, где $Z \subseteq X$, $x = X_0 \cup \left(\bigcup_{l=1}^{n_r} Z_{i_l}^{out} \right)$; $f_{i_l} \in F_r$; $i_l \in \overline{1,n_f}$; $l = \overline{1,n_r}$, n_r – число выполненных операций схемы s, $y = \left(Y_0 \cup \left(\bigcup_{l=1}^{n_w} Z_{j_k}^{in} \right) \right) \Big/ x$; $f_{j_k} \in F_w$, F_w – множество невыполненных операций схемы s, $j_k \in \overline{1,n_f}$, $k = \overline{1,n_w}$, n_w – число невыполненных операций схемы s.

Сценарии обработки отказов

Виды, признаки и причины отказов объектов вычислительной среды с различной степенью их детализации описаны в [1]. Мы рассматриваем следующие объекты среды: узлы, агенты и модули.

Соответственно, мы учитываем следующие отказы: отказ узла (узел находится в нерабочем состоянии); отказ агента (агент не отвечает в течение установленного периода времени); отказ модуля (аварийное завершение выполнения модуля). Эти отказы обобщают различные причины и признаки неисправностей рассматриваемых объектов. Каждый отказ приводит к необходимости перезапуска модуля схемы решения задачи или выполнения другого модуля.

Рисунок иллюстрирует сценарии обработки перечисленных выше отказов.

Модуль может быть выполнен другим агентом в случае всех трех отказов. Кроме того, при отказе узла модуль может быть выполнен также в резервном узле. Дополнительно, при отказе модуля выполнявшейся схемы решения задачи может быть выполнен модуль эквивалентной схемы решения задачи.

Есть три возможных варианта назначения агента для выполнения модуля:

– централизованное (директивное) назначение модуля другому агенту, имеющему возможность выполнить данный модуль, координатором выполнения схемы решения задачи;

Сценарии обработки отказов

– децентрализованное назначение модуля путем проведения торгов между агентами, которые могут выполнить данный модуль;

– децентрализованное назначение модулей остаточной схемы решения задачи путем проведения торгов между агентами, которые могут принять участие в выполнении модулей остаточной схемы.

Состав виртуального сообщества агентов, выполняющих схему решения задачи, обновляется после назначения новых агентов для выполнения модулей. Выделение ресурсов агентами осуществляется с учетом различий вычислительных характеристик узлов Grid при реализации многоуровневого параллелизма алгоритма решения задачи.

Алгоритм

Пусть Z^* – это множество простых и составных логических параметров, принимающих значения из множества {0, 1, θ}, $Z^* \subset Z$. Составной параметр реализует логическое выражение, сформированное из простых параметров и логических операторов. Составной параметр не определен, если не определен хотя бы один из его простых параметров. Обозначим через A и PR множество действий по обработке отказов и множество продукций, определяющих правила выбора действий: $pr_i : u_i^{pre}; z_j \Rightarrow a_k; u_i^{post}$, где предусловие продукции определяется как

$$u_i^{pre} = \begin{cases} 0, \text{если } z_j = \theta, \\ 1, \text{если } z_j = 0 \vee z_j = 1, \end{cases}$$

$z_j \Rightarrow a_k$ – ядро продукции, интерпретируемое как выбор действия $a_k \in A$ по обработке отказа, $z_j \in Z^*$, $u_i^{post} : A \to Z^*$ – постусловие продукции. Каждая продукция имеет приоритет.

Связи между продукциями и логическими параметрами в левых частях их ядер представлены булевой матрицей B, размерности $n_p \times n_z$, где n_p – число продукций; n_z – число логических параметров. Элемент матрицы $b_{i,j} = 1$ означает, что продукция pr_i использует параметр z_j. Связи между продукциями и действиями представлены булевой матрицей R, размерности $n_p \times n_a$, где n_a – число действий. Элемент матрицы $r_{i,k} = 1$ означает, что продукция pr_i описывает правило выполнения действия a_k. Зависимости между действиями представим булевой матрицей D размерности $n_a \times n_a$. Элемент матрицы $d_{k_1,k_2} = 1$ означает, что действие a_{k_1} зависит от действия a_{k_2}.

Адаптивный алгоритм восстановления вычислительного процесса включает следующие основные этапы.

1. Обработка предусловий u_i^{pre} продукций pr_i, $i = \overline{1,n_p}$.

2. Если $\bigvee_{i=1}^{n_p} u_i^{pre} = 0$, то задача неразрешима. Переход на шаг 10.

3. Формирование списка продукций $pr_{i_1}, pr_{i_2}, ..., pr_{i_q}$, ядра которых могут быть выполнены: $u_{i_h}^{pre} = 1$, $i_h \in \overline{1,n_p}$, $h = \overline{1,q}$, $1 \le q \le n_p$.

4. Вычисление левых частей $z_{j_1}, z_{j_2}, ..., z_{j_q}$ ядер продукций $pr_{i_1}, pr_{i_2}, ..., pr_{i_q}$: $b_{i_h,j_h} = 1$, $\forall i_h \in \overline{1,n_p}$, $\forall j_h \in \overline{1,n_z}$.

5. Если $\bigvee_{h=1}^{q} z_{j_h} = 0$, то задача неразрешима. Переход на шаг 10.

6. Выбор действия $a_k : r_{n,k} = 1$, где $n = \underset{h=\overline{1,q}}{\operatorname{argmax}}\ \hat{p}_{i_h}$, $\hat{p}_{i_h}$ – приоритет продукции p_{i_h}.

7. Выполнение действия a_k и обработка постусловия продукции p_n.

8. Если $\bigvee_{\substack{e=1 \\ e\neq k}}^{n_a} d_{k,e} = 0$, то задача решена. Переход на шаг 10.

9. Конкретизация вычислительной модели: $A = A\backslash\{a_k\}$, $PR = PR\backslash\{p_n\}$. Переход на шаг 1.

10. Завершение работы алгоритма.

Обработка предусловий продукций обеспечивает адаптацию работы алгоритма к текущему состоянию Grid и выбор наиболее подходящего сценария действий по устранению отказа. Алгоритмы, применяемые при выполнении децентрализованного назначения модулей, идентификации отказов и резервирования узлов, рассмотрены в [3, 5, 7].

Модель функционирования агентов

Алгоритмы функционирования агентов разработаны на основе конечно-автоматной модели в соответствии со спецификой действий, выполняемых этими агентами в системе. Модель имеет следующий вид

$$M_a = \langle STS, sts_0, ACT, MES, SLT, g_l \rangle,$$

где STS – множество состояний агента; $sts_0 \in STS$ – начальное состояние агента; ACT – множество действий, совершаемых агентом, которое включает подмножество локальных действий $ACT_{locl} \subset ACT$, а также подмножества коммуникационных взаимодействий с другими агентами: отправки сообщений $ACT_{send} \subset ACT$ и приема сообщений $ACT_{recv} \subset ACT$; MES – множество входных и выходных сообщений агента, SLT – система логического времени для датировки событий мультиагентной системы, g_l – логическая функция переходов из одного состояния в другое в результате действий, совершаемых агентом.

Система логического времени SLT представлена следующей структурой:

$$SLT = \langle T, T_m, g_t, g_m, g_r \rangle,$$

где T – область значений логического времени; T_m – область значений временных маркеров датировки сообщений, $T_m \subseteq T$; g_t – функция логического времени; g_m – функция датировки сообщений; g_r – функция сравнения значений логического времени. Система логического времени использует векторные логические часы, в которых число компонент вектора времени равно числу агентов виртуального сообщества.

Реализация агентов осуществляется с помощью Java Agent DEvelopment framework (JADE) [2]. Сообщения создаются с помощью языка Agent Communication Language [2]. Очередь сообщений агента в JADE формируется в порядке поступления сообщений на физическом уровне. Этот порядок может не соответствовать логической обусловленности событий мультиагентной системы. Причинами такого несоответствия могут быть как рассинхронизация физических часов агентов, так и задержка передачи сообщений в коммуникационных каналах связи.

Система логического времени позволяет определить отношение частичного порядка на множестве событий мультиагентной системы с учетом их обусловленности. Использование функции датировки сообщений обеспечивает их обработку в установленной логической последовательности, а не в произвольном порядке поступления их в очередь.

Для обеспечения надежности системы передачи сообщений мы используем протокол с двумя сообщениями [8], который может приводить к дублированию, но не допускает потери информации, так как все сообщения содержат временные маркеры датировки сообщений и идентификаторы виртуальных сообществ агентов.

Заключение

В статье представлен адаптивный мультиагентный алгоритм, предназначенный для перераспределения вычислительных ресурсов Grid при возобновлении процесса решения задачи в рамках модульной системы программирования. Функционирование данного алгоритма базируется на методах конкретизирующего программирования для построения и выполнения остаточной схемы решения задачи. Предложена конечно-автоматная модель поведения агента, использующая систему логического времени и надежный протокол обмена сообщениями.

Исследование выполнено при финансовой поддержке РФФИ, проекты № 15-29-07955-офи_м и № 16-07-00931-а, а также при частичной финансовой поддержке Совета по грантам Президента Российской Федерации для государственной поддержки ведущих научных школ Российской Федерации (НШ-8081.2016.9).

Список литературы

1. Balaji P., Buntinas D., Kimpe D. Fault Tolerance Techniques for Scalable Computing / Wang L., Zomaya A.Y., Khan S.U. // Scalable Computing and Communications // Theory and Practice. – Hoboken: Wiley-IEEE Press, 2013. – P. 212–245.

2. Bellifemine F., Bergenti F., Caire G., Poggi A. Jade – A Java Agent Development Framework / R. Bordini et al // Multiagent Systems, Artificial Societies, And Simulated Organizations: Multi-Agent Programming. – Berlin: Springer, 2006. – Vol. 15. – P. 125–147.

3. Bogdanova V.G., Bychkov I.V., Korsukov A.S., Oparin G.A., Feoktistov A.G. Multiagent Approach to Controlling Distributed Computing in a Cluster Grid System // Journal of Computer and Systems Sciences International. – 2014. – Vol. 53, № 5. – P. 713–722.

4. Bychkov I.V., Oparin G.A., Feoktistov A.G., Bogdanova V.G., Pashinin A.A. Service-oriented multiagent control of distributed computations // Automation and Remote Control. – 2015. – Vol. 76, № 11. – P. 2000–2010.

5. Bychkov I.V., Oparin G.A., Feoktistov A.G., Sidorov I.A., Bogdanova V.G., Gorsky S.A. Multiagent control of computational systems on the basis of meta-monitoring and imitational simulation // Optoelectronics, Instrumentation and Data Processing. – 2016. – Vol. 52, № 2. – P. 107–112.

6. Ershov A.P. On mixed computation: informal account of the strict and polyvariant computation schemes / M. Broy // Control flow and data flow: concepts of distributed programming. – Berlin a.o.: Springer-Verlag, 1985. – P. 107–120.

7. Feoktistov A.G., Sidorov I.A. Logical-Probabilistic Analysis of Distributed Computing Reliability // In proceedings of the 39th International Convention on information and communication technology, electronics and microelectronics (MIPRO-2016). – Riejka: Croatian Society for Information and Communication Technology, Electronics and Microelectronics, 2016. – P. 247–252.

8. Tel G. Introduction to Distributed Algorithms. – Cambridge: Cambridge University Press, 2000. – 596 p.

9. Mishra M.K., Patel Y.S., Rout Y., Mund G.B. A Survey on Scheduling Heuristics in Grid Computing Environment // International Journal of Modern Education and Computer Science. – 2014. – Vol. 6, № 10. – P. 57–83.

10. Qureshi M.B., Dehnavi M.M., Min-Allah N., Qureshi M.S., Hussain H., Rentifis I., Tziritas N., Loukopoulos T., Khan S.U., Xu C.Z., Zomaya A.Y. Survey on Grid Resource Allocation Mechanisms // Journal of Grid Computing. – 2014. – Vol. 12, № 2. – P. 399–441.