Исследование метода журнала событий для организации отказоустойчивых и самобалансирующихся вычислений в гибридной среде

С.В. Востокин

Самарский университет, Самара, Россия

В статье представлено экспериментальное исследование приложения для гибридных сред, в котором применен новый метод синхронизации состояний его распределенных компонентов через отправку специальных заданий на узел журнала событий. Метод отличается от традиционных грид-приложений, где один узел управления создает множество автономных заданий для выполнения в гриде. Он позволяет использовать удобную для программиста модель SPMD. Также он успешно решает проблемы отказоустойчивости и балансировки нагрузки, обеспечивая ускорение с небольшими накладными расходами.

ВВЕДЕНИЕ

В связи с ростом объема вычислений в области обработки больших данных и искусственного интеллекта, а также при решении традиционных задач численного моделирования, возникает потребность в программах, которые можно развертывать и запускать в гибридных средах, состоящих из произвольных набор сетевых вычислительных ресурсов. Хорошими примерами таких ресурсов являются (а) добровольные компьютеры, такие как платформа BOINC [1] или другие добровольные проекты распределенных вычислений; (б) простаивающие корпоративные компьютеры, которые потенциально доступны по сети для решения производственных задач; (в) простаивающие вычислительные узлы высокопроизводительных суперкомпьютерных или кластерных систем [2, 3]; (г) бесплатные или недорогие спотовые виртуальные машины от облачных провайдеров [4].

Использование гибридных сред позволяет снизить стоимость и добиться высокой производительности. Но ключевыми проблемами при программировании приложений для гибридных сред являются отказоустойчивость и балансировка нагрузки. Специфика гибридной среды приводит к тому, что данные проблемы должно решать само приложение, а не его вычислительная среда. Например, мы не сможем использовать аппаратную виртуализацию в облаке. Компоненту приложения в гибридной среде необходимо самостоятельно быстро начать вычисления на вновь подключенном ресурсе. При этом внезапное или плановое отключение ресурса с развернутым на нем компонентом приложения не должно приводить к сбою всего приложения. В результате требуется специальный дизайн приложения.

В статье представлено экспериментальное исследование дизайна приложения для гибридной среды выполнения. Мы ограничимся выполнением множества независимых задач, чтобы продемонстрировать применимость метода. Сначала мы обсудим традиционное многозадачное приложение и исследуемую его альтернативную архитектуру, подчеркнув преимущества последней. Во-вторых, мы представляем модельную задачу и результаты ее моделирования и

нагрузочного тестирования. В заключении делаем выводы по экспериментам и дальнейшей работе.

ВЫЧИСЛИТЕЛЬНАЯ АРХИТЕКТУРА

На рисунке (см. рис. 1) показана традиционная архитектура распределенного многозадачного приложения. На рисунке облачная платформа Everest [5] используется в качестве промежуточного программного обеспечения для распределенного приложения. В традиционном дизайне приложений один выделенный узел управляет вычислениями на других рабочих узлах. Рабочие узлы имеют агентов платформы Everest. Узел оркестратора взаимодействует с платформой, отправляя задания через протокол REST.

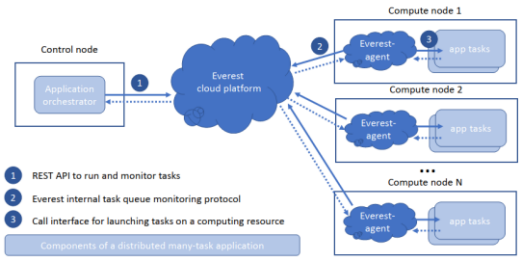

Рис. 1. Традиционная архитектура распределенного приложения (на основе промежуточного программного обеспечения Everest)

Мы предлагаем инвертировать вычислительную архитектуру (см. рис. 1), что дает нам новую архитектуру (см. рис. 2).

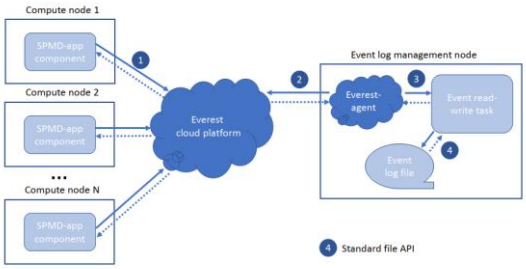

Рис. 2. Экспериментальная архитектура с журналом событий (на базе промежуточного программного обеспечения Everest)

Компоненты приложения располагаются на отдельных узлах. Каждый компонент выполняет один и тот же код, более того, он поддерживает то же состояние вычислений, что и другие компоненты, реализуя модель программирования SPMD. Компоненты периодически взаимодействуют через платформу Everest по протоколу REST. Взаимодействие основано на отправке специального задания на платформу Everest. Суть задания заключается в том, чтобы записать локальное изменение состояния в журнал событий в качестве входных данных и получить на выходе изменения состояний от других узлов. Задание выполняется на узле управления журналом событий (см. рис.2). Только на этом узле должен быть установлен агент Everest. Компонент задачи чтения/записи событий используется для доступа к файлу журнала событий. Вычислительная архитектура, показанная на рис. 2, имеет следующие

преимущества. С точки зрения конфигурации вычислений это позволяет быстро развернуть компоненты приложения СПМД на невыделенных вычислительных ресурсах, поскольку установка агента ресурсов не требуется. Кроме того, архитектура отказоустойчива и позволяет переносить код без остановки вычислений, что достигается путем копирования файла журнала событий с одного узла управления журналом событий на другой. При этом прикладной компонент СПМД может быть запущен или остановлен на любом узле в любой момент выполнения. Компоненты приложения следуют модели пилотного задания [6].

ПРИМЕР ПРИМЕНЕНИЯ И РЕЗУЛЬТАТЫ ЭКСПЕРИМЕНТА

Постановка задачи следующая. У нас есть произвольное количество независимых задач. В экспериментах оно варьируется от 10 до 50 с шагом 10. Задачи должны решаться в 10 процессах или вычислительных узлах. Для каждого процесса определен индивидуальный случайный порядок, в котором он будет решать задачи. Определено время расчета одной задачи. В экспериментах оно меняется от 10 до 50 секунд с шагом 10 секунд.

Реализация гарантирует следующие условия: (а) каждая задача обрабатывается хотя бы одним процессом; (б) все процессы согласовывают общий порядок, в котором будут получены решения задач.

В ходе эксперимента была произведена оценка ускорения расчета на 10 процессах при отсутствии коммуникационных накладных расходов при моделировании и при наличии коммуникационных накладных расходов при нагрузочном тестировании. Результаты эксперимента представлены в таблице 1.

Таблица 1. Результаты эксперимента: ускорение при использовании 10 рабочих процессов

Длительность задачи, сек	**Количество задач, шт.**				
	10	**20**	**30**	**40**	**50**
при отсутствии накладных расходов на связь при моделировании					
10 – 50	3.33333	4.0	5.0	5.71429	5,55556
при наличии коммуникационных издержек при нагрузочном тестировании					
10	1,23553	2,02957	3,34938	2,84712	3.19064
20	2,23945	3,44074	3,73296	4,97036	5,42627
30	3.0209	3,62881	4.55871	5.20853	5.09511
40	3,04433	3,71611	4,66365	5.29505	5.21234
50	3.1375	3,77058	4,72567	5.40605	5.28163

Имитационный эксперимент реализован с помощью Templet SDK [7]. Язык программирования — C++ с интерпретатором Cling. Средой разработки является блокнот JupyterLab , развертывание в облаке осуществляется с помощью Binder [8]. Код нагрузочного тестирования реализован на C++ с использованием компилятора GCC. Промежуточное ПО – платформа Everest ИППИ РАН [5], libcurl реализует связь с платформой.

Развертывание осуществлялось на виртуальной машине в OVHcloud с помощью сервиса Binder. Процесс журнала событий и вычислительные процессы были развернуты на одной виртуальной машине, чтобы создать максимальную нагрузку на систему связи.

Эксперименты [9] показывают, что метод вычислений на основе журнала событий позволяет успешно решать задачи отказоустойчивости, балансировки

нагрузки, обеспечивает ускорение вычислений. Но метод имеет ограничение в форме избыточного объема вычислений. Оно не существенно, учитывая доступность и низкую стоимость вычислительных ресурсов в гибридном облаке.

ВЫВОДЫ

Мы представили метод журнала событий и экспериментальное исследование соответствующего примера приложения, которое показывает его применимость для вычислений в гибридной среде. В дальнейшей работе мы планируем адаптировать метод для приложений с динамически генерируемым набором зависимых задач и устранить ограничения по ускорению в текущей реализации.

ИСПОЛЬЗОВАННАЯ ЛИТЕРАТУРА

1. *Андерсон Д.П.* BOINC: платформа для добровольных вычислений // Журнал Grid Computing. – 2020. – Т. 18, №1. – С. 99-122.
2. *Де К. и др.* Интеграция системы управления рабочей нагрузкой PanDA с суперкомпьютером Titan в OLCF // Физический журнал: Серия конференций. – Издательство ИОП, 2015. – Т. 664, вып. 9. – С. 092020.
3. *Кореньков В.В., Кондратьев А.О., Бондяков А.С.* Технология взаимодействия клиента jAliEn и центральных сервисов ALICE // Современные информационные технологии и IT-образование. – 2019. – Т. 15, №3. – С. 602-610.
4. *Варшней П., Симхан Ю.* AutoBoT : Гибкое и экономичное планирование пакета задач на локальных виртуальных машинах // Транзакции IEEE в параллельных и распределенных системах. – 2018. – Т. 30, №7. – С. 1512-1527.
5. *Волков С., Сухорослов О.* Упрощение использования облаков для научных вычислений с помощью Everest // Procedia Computer Science. – 2017. – Т. 119. – С. 112-120.
6. *Турилли М., Санткроос М., Джа С.* Комплексный взгляд на системы пилотных заданий // ACM Computing Surveys (CSUR). – 2018. – Т. 51, №2. – С. 1-32.
7. *Востокин С., Попов С., Сухорослов О.* Опыт организации гибкого доступа к удаленным вычислительным ресурсам из среды JupyterLab с использованием технологий проектов Everest и Templet //Материалы семинара CEUR. – 2021. – Т. 3041. – С. 558-561.
8. *Раган-Келли Б. и др.* Binder 2.0 — Воспроизводимые, интерактивные, совместно используемые среды для науки в больших масштабах // Материалы 17-й конференции Python in Science. – Ф. Акичи , Д. Липпа, Д. Нидерхут и М. Пейсер, ред., 2018. – С. 113-120.
9. Результаты экспериментов [Электронный ресурс]. Доступно по адресу: https://github.com/the-templet-project/templet/tree/master/samples/blchsym (по состоянию на 2 сентября 2023 г.).