Методы обеспечения отказоустойчивости, основанные на координированном сохранении контрольных точек*

А.А. Бондаренко[1], П.А. Ляхов[2], М.В. Якобовский[1]

ИПМ им. М.В. Келдыша РАН[1], МФТИ(ГУ) [2]

В области высокопроизводительных вычислений отказоустойчивость становится одной из главных проблем из-за увеличения отказов в оборудовании. Наиболее перспективное решение состоит в предоставлении пользователю возможности обрабатывать отказ в своем приложении на уровне MPI, например, с помощью дополнительных функций ULFM. В данной работе рассматриваются методы координированного сохранения контрольных точек на уровне пользователя: сохранение в распределенную файловую систему, многоуровневый метод сохранения, метод массового дублирования на локальные устройства хранения. Дается оценка применимости этих методов сохранения для организации отказоустойчивых вычислений.

Ключевые слова: параллельное программирование, MPI, расширение ULFM, контрольные точки, координированное сохранение, отказоустойчивые вычисления.

1. Введение

Создание суперкомпьютеров экзафлопсного уровня производительности сопряжено с необходимостью решения ряда задач, среди которых важное место занимает обеспечение возможности выполнения расчетов на вычислительных системах, несмотря на отказ ряда задействованных в расчете вычислительных узлов. Современные суперкомпьютеры состоят из объединенного сетью большого количества вычислительных узлов, каждый из которых содержит множество элементов, в том числе: процессоры, различные ускорители, модули оперативной памяти, сетевые контроллеры, диски и другие. Кратное увеличение количества компонент системы естественным образом ведет к экспоненциальному увеличению вероятности отказа одного из них.

Уже сейчас для вычислительной системы время наработки на отказ сравнимо со временем проведения одного расчета, в связи, с чем основной стратегией является периодическое сохранение промежуточных результатов (глобальной контрольной точки) на надежное устройство хранения. Однако, в ближайшей перспективе время между отказами компонент суперкомпьютера может стать меньше не только времени проведения всего расчета, но и времени, требуемого на загрузку контрольной точки, проведения части вычислений и сохранения следующей контрольной точки. Таким образом, актуальным становится вопрос об обеспечении самой возможности осуществления длительного расчета на перспективных вычислительных системах.

В литературе [1] широко представлены различные техники обеспечения отказоустойчивости, основными из которых являются методы, основанные на контрольных точках, различных протоколах восстановления и их модификации. Создание контрольных точек связано с накладными расходами, возникающими из-за интенсивного использования узлов ввода-вывода системы при записи на общее устройство хранения, поэтому среднее время сохранения контрольных точек может быть весьма значительным. Например, характерное время сохранения контрольных точек на системном уровне (осуществляется копирование всей оперативной памяти), основанное на библиотеке BLCR [2], для некоторых систем близких к Петафлопсному уровню составляет от 20 до 30 минут, а для будущих Экзафлопсных вычислительных систем использование подобной техники выглядит неприемлемо [3]. Эти и другие проблемы побудили исследователей к разработкам новых методов, в том числе повышающих масштабируемость техник сохранения контрольных точек.

* Работа выполнена при поддержке Российского фонда фундаментальных исследований по гранту 17-07-01604 А

Отметим, что ключевым решением для уменьшения накладных расходов при сохранении контрольных точек и восстановлении после отказа состоит в том, чтобы предоставить пользователю возможность самому определять объем контрольной точки, а в некоторых случаях и реализовать в приложении различные техники отказоустойчивости. Проблема усугубляется тем, что в текущем стандарте MPI 3.1 отсутствуют механизмы позволяющие работать с отказами. На данный момент ULFM – расширение стандарта MPI, позволяющее реализовывать методы обеспечения отказоустойчивости на уровне пользователя, находится в разработке [4]. Программные реализации предварительных версий стандарта MPI с расширением ULFM были представлены на конференциях SC'14, SC'15, SC'16 и доступны [4].

В данной работе рассматриваются методы координированного сохранения контрольных точек на уровне пользователя: сохранение в распределенную файловую систему, многоуровневый метод сохранения, метод массового дублирования на локальные устройства хранения. Дается оценка применимости этих методов для организации отказоустойчивых вычислений. Для достижения поставленной цели рассмотрена теоретическая модель [5] выполнения параллельных вычислений и приведены результаты вычислительного эксперимента, в котором тестовые программы, реализованные с помощью ULFM-функций, запускались на вычислительном кластере [6].

2. Методы сохранения контрольных точек

Опишем методы координированного сохранения контрольных точек на уровне пользователя: многоуровневый метод сохранения [5] и метод массового дублирования на локальные устройства хранения [7].

2.1 Многоуровневая схема сохранения и сохранение контрольных точек в распределенную файловую систему

В статье [5] предлагается способ определения оптимального количества контрольных точек на уровне в случае многоуровневого координированного сохранения. В нашем случае мы рассматриваем двухуровневую систему сохранение контрольных точек: в память соседнего процесса (уровень 1) и в распределённую файловую систему (уровень 2). Примем, что произошел отказ первого уровня, если для восстановления расчетов достаточно данных сохранённых в оперативной памяти соседнего процесса. Аналогично отказ называем отказом второго уровня, если для восстановления расчетов не достаточно данных сохраненных в оперативную память соседа, а необходимо прочесть эти данные из распределенной файловой системы. Отказы второго уровня (и более высоких уровней) происходят, когда отказывает сразу два процесса (несколько процессов).

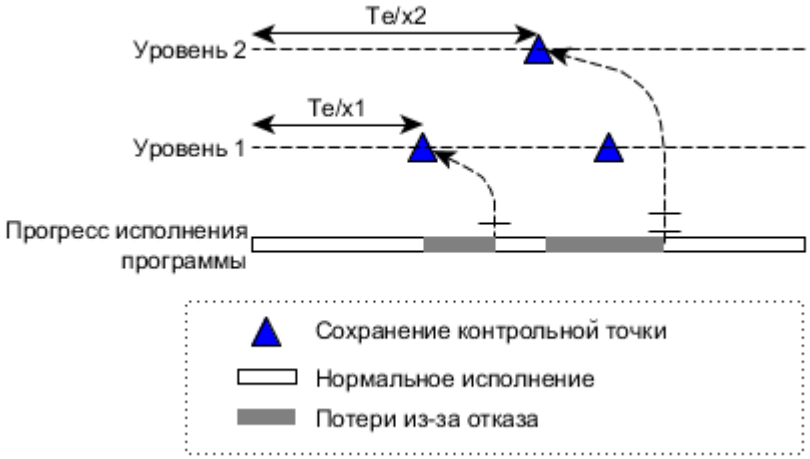

Рис. 1. Исполнение HPC-приложения с отказами

В соответствии со статьёй [5] полное время работы программы можно оценить формулой:

$$T_{final} = T_e + C_1(x_1 - 1) + C_2(x_2 - 1) + \left(\frac{T_e}{2x_1} + D + R_1\right)n_1 + \\ + \left(\frac{T_e + C_1 x_1}{2x_2} + D + R_2\right)n_2$$

Для записи формулы и далее используются обозначения:

- T_{final} – суммарное время работы программы, с учётом наличия отказов в системе;
- T_e – время на полезные вычисления в ходе программы;
- C_i – накладные расходы на сохранение контрольной точки на *i*-ом уровне;
- x_i – искомое оптимальное число периодов между контрольными точками на *i*-ом уровне;
- $x_i^{(k)}$ – число периодов между контрольными точками на *i*-ом уровне в *k*-ом приближении;
- n_i – ожидаемое число отказов на *i*-ом уровне за время работы программы;
- D – время на восстановление работоспособности MPI;
- R_i – время на восстановление из контрольной точки.

В соответствии с [5] для минимизации времени исполнения по x_i достаточно найти решение системы уравнений .

$$\frac{\partial T_{final}}{\partial x_i} = 0$$

Для двух уровней сохранения система уравнений принимает вид (1).

$$\begin{cases} C_2 = \dfrac{n_2}{2x_2^2}(T_e + C_0 x_0) \\ C_1 = \dfrac{n_1}{2x_1^2}T_e - \dfrac{C_1}{2}\dfrac{n_2}{x_2} \end{cases} \quad (1)$$

В статье [5] предлагается решать данную систему численно. В качестве начального приближения числа промежутков между контрольными точками берутся

$$x_1^{(0)} = \sqrt{\frac{n_1 T_e}{2C_1}}, \quad x_2^{(0)} = \sqrt{\frac{n_2 T_e}{2C_2}},$$

после чего последовательно вычисляются следующие приближения, связанные с предыдущими по формуле (2).

$$x_1^{(k+1)} = \sqrt{\frac{n_1 T_e}{C_1\left(2 + \dfrac{n_2}{x_2^{(k)}}\right)}}, \qquad x_2^{(k+1)} = \sqrt{\frac{n_2\left(T_e + C_1 x_1^{(k)}\right)}{2C_2}} \quad (2)$$

Таким образом, число контрольных точек на *i*-ом уровне равно $x_i^{(k)} - 1$, а соответствующий период между сохранениями контрольных точек равен $\frac{T_e}{x_i^{(k)}}$. Вычисления продолжаются до тех пор, пока не выполнится условие $\max_i \left|x_i^{(k+1)} - x_i^{(k)}\right| < \varepsilon$, где $\varepsilon = 10^{-6}$, что в среднем достигается за 30 итераций.

Таким образом, входными параметрами системы уравнений являются n_i, T_e и C_i. Последние два определяются на стадии запуска копии программы. Каждый из рабочих процессов замеряет время, которое он тратит на сохранение контрольной точки на соответствующий уровень, после чего для каждого уровня итоговое C_i берётся как среднее арифметическое этих значений. После этого производится вычисление нескольких шагов по времени, замеряется время работы на этом числе шагов, откуда получаем среднее время вычисления одного шага по времени. Умножая это число на общее число шагов по времени в исходной программе, определяется T_e – полное время работы программы без применения средств обеспечения отказоустойчивости.

Выбор диапазона значений n_i основывается на следующих соображениях. Пусть $T_e \sim$ 1 день, авторы [8] ожидают, что среднее время между отказами (MTBF) будет составлять менее часа. Таким образом, при MTBF = {0.5, 1, 2} часа соответствующее число отказов будет равно 48, 24, 12,

Частота сохранение контрольных точек в распределенную файловую систему определяется аналогичным образом, когда в формуле (1) и (2) искусственно полагают $n_1 = 0$, а советующее значение $n_2 \in \{12,24,48\}$.

2.3 Метод массового дублирования на локальные устройства хранения

Пусть MPI-процессы сохраняют локальные контрольные точки в память своих вычислительных узлов. Тогда, при отказе хотя бы одного вычислительного узла, MPI-процессы, запущенные на других узлах, не смогут получить доступ к локальным контрольным точкам, расположенным в памяти отказавшего. Таким образом, восстановление процесса вычислений будет невозможно. Для выхода из такого положения следует обеспечить избыточность хранения локальных контрольных точек в системе за счет их дублирования в памяти различных вычислительных узлов. В соответствующей схеме записи для каждого MPI-процесса определяются номера вычислительных узлов, в память которых должны быть сохранены копии локальной контрольной точки.

Для описания схемы записи локальных контрольных точек будем использовать следующие параметры: N – число узлов в системе; MNF – максимально допустимое число отказов в системе; глубина хранения SD (storage depth) – это количество итераций сохранения, в которые каждая локальная контрольная точка доступна для чтения; коэффициент дублирования DF (duplication factor) – это количество узлов хранящих копию данной локальной контрольной точки, отличных от данного вычислительного узла. Цель данной схемы сохранения состоит в максимизации минимального числа узлов, повреждение которых приведёт к невозможности восстановления с последних SD итераций сохранения.

Предполагаем, что на один вычислительный узел приходится один MPI-процесс. Данное предположение не снижает общности последующих рассуждений. При запуске на одном узле нескольких MPI-процессов необходимо всем MPI-процессам с одного узла осуществлять запись и чтение копий локальных контрольных точек в память вычислительных узлов, определяемых формулой (1). В этом случае в системе будет доступна согласованная глобальная контрольная точка, если число отказов узлов не превосходит значение MNF.

Используется координированный протокол сохранение, под локальными устройствами памяти понимаем оперативную память, в общем случае ими также могут быть SSD/HDD диски.

Введем нумерацию итераций сохранения контрольных точек $k = 0,1,2, \ldots$. На одной итерации сохранения MPI-процесс с каждого *i*–ого вычислительного узла:

- сохраняет свою локальную контрольную точку в память этого узла;
- осуществляет запись DF копий своей локальной контрольной точки в память вычислительных узлов, определяемых формулой (1).

Введем параметр $j \in \{1, 2, \ldots, DF\}$ для обозначения номера передаваемой копии локальной контрольной точки с *i*–ого узла. Тогда на *k*–ой итерации сохранения, запись *j*-ой копии с *i*–ого узла должна быть осуществлена в память вычислительного узла с номером, определяемым формулой:

$$receiver(i,j,k) = \left[i + j \cdot DF^{k \bmod SD} + k \bmod SD\right] \bmod N$$

В данной схеме объем сохраняемой на каждом узле информации можно оценить величиной $V = V_0 \cdot SD \cdot (DF + 1)$, где V_0 – средний объем локальных контрольных точек.

Пусть в вычислительной системе достаточно много узлов $N \geq DF^{SD} + SD$ и число отказавших узлов не превосходит

$$MNF = (DF - 1) \cdot SD + 1$$

Тогда, после первых SD итерации сохранения, в системе будут доступны локальные контрольные точки MPI-процессов со всех узлов одной из последних SD итераций сохранения, то есть согласованная глобальная контрольная точка. Таким образом, используя предлагаемую схему,

можно восстановить процесс вычислений, если количество отказов в системе не превышает значения MNF.

Для массового дублирования на локальные устройства хранения определяется свое значения параметра C_1, при этом частота сохранение контрольных точек определяется по формулам (1) и (2), где искусственно полагают $n_2 = 0$, а советующее значение $n_1 \in \{12,24,48\}$.

3. Вычислительный эксперимент

Обработка исключений и реализация различных техник восстановления осуществляется, в основном за счет функций [4]:

- MPI_COMM_REVOKE — прекращает все текущие нелокальные операции на коммуникаторе и отмечает коммуникатор в качестве аннулированного. Все последующие вызовы нелокальных функций, связанные с этим коммуникатором, должны завершаться значением MPI_ERR_REVOKED, за исключением функций MPI_COMM_SHRINK и MPI_COMM_AGREE;
- MPI_COMM_SHRINK — создает на основе существующего коммуникатора новый, не содержащий отказавшие процессы;
- MPI_COMM_FAILURE_GET_ACKED — возвращает группу, состоящую из процессов, которые были определены как отказавшие к моменту последнего вызова функции MPI_COMM_FAILURE_ACK;
- MPI_COMM_AGREE — согласовывает значение булевой переменной, если нет отказавших MPI-процессов в коммуникаторе или возвращает исключение о наличии отказа всем не отказавшим процессам в коммуникаторе.

Для моделирования отказа отдельного процесса в системе используется системная библиотека signal. В заранее определённый момент времени, выбранный процесс (или процессы) совершает вызов функции raise(SIGKILL), что приводит к его экстренному завершению, без выполнения какого-либо последующего кода. Для того чтобы определить эти моменты времени, время работы программы делится на число отказов на соответствующем уровне (что устанавливает время между отказами на этом уровне). Сами отказы происходят равномерно, через равные промежутки времени, соответствующие полученным временам между отказами.

В данной работе были разработаны три программы, реализующие параллельный алгоритм решения задачи о распространении тепла в однородном стержне. Отличие для каждой программы заключается в методе сохранения контрольных точек, а восстановление после отказа для программ существенно не отличается и состоит в загрузке всем процессам последней согласованной глобальной контрольной точки. В первой программе сохранение контрольных точек осуществляется в оперативную память вычислительных узлов и распределенную файловую систему согласно многоуровневому методу сохранения, во второй программе сохранение только в распределенную файловую систему, а в третье программе запись контрольных точек осуществлялась согласно методу массового дублирования на локальные устройства хранения.

В случае возникновения отказа, MPI-процессы, отмеченные как отказавшие, выводятся из расчетного поля. Для восполнения исходного количества процессов в рабочем поле осуществляют порождение новых (`MPI_Comm_spawn`) или их берут из группы заранее зарезервированных процессов. Количество резервных MPI-процессов является параметром для программы, и должно определяться пользователем исходя из предположений о возможном количестве отказов во время выполнения программы. Такое разделение на рабочие и резервные MPI-процессы, связано с тем, что для большинства алгоритмов, описывающих решение задач математической физики, затратными являются процедуры перераспределения работы на меньшее число MPI-процессов, по сравнению с начальными условиями запуска программы.

Отметим, что для реализации отказа первого уровня осуществлялся вызов raise(SIGKILL) для одного mpi-процесса, а для отказа второго уровня на двух соседних mpi-процессах. Для второй и третьей программы при сохранении контрольных точек только в оперативную память или только в распределенную файловую систему было принято решение рассматривать все суммарные отказы, как отказы второго уровня, в связи с чем для каждой сетки они принимают одно значение при соответствующем суммарном числе отказов.

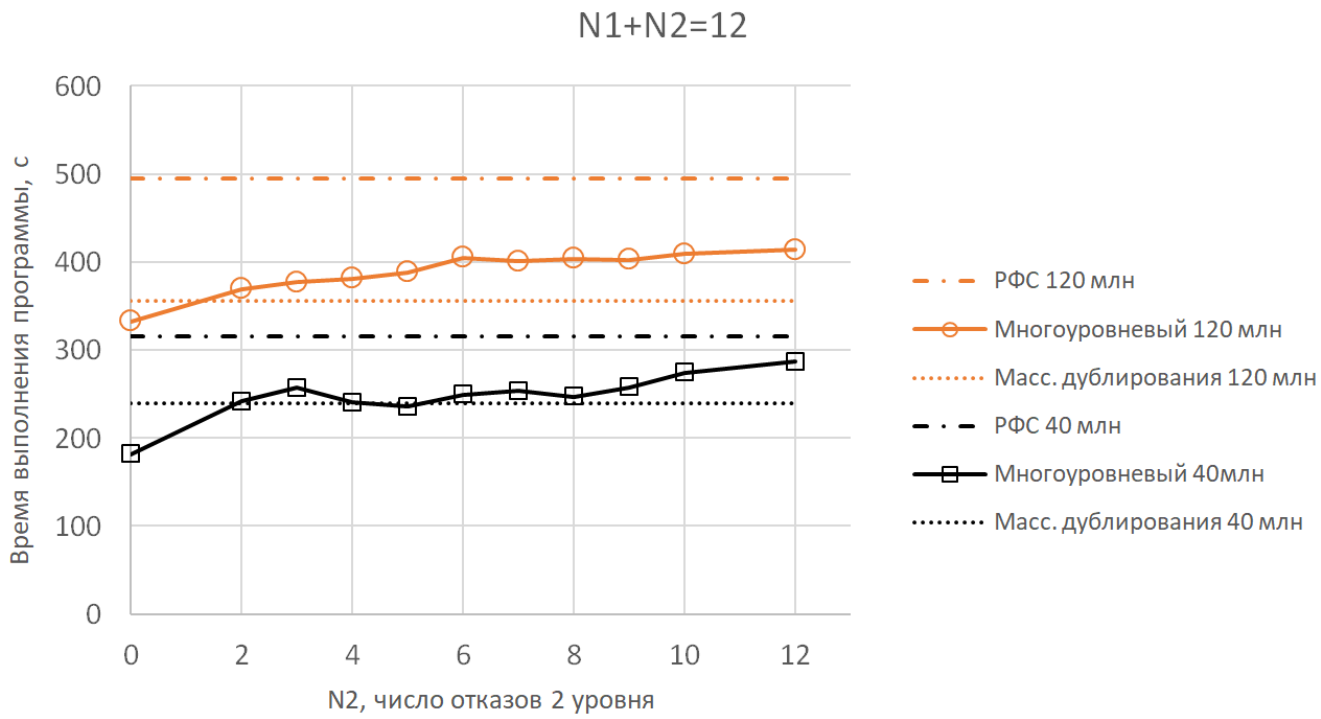

Рис.2 Время выполнения трех параллельных программ на сетках 40 и 120 млн. узлов при наличии отказов разных уровней, суммарное число которых равно 12.

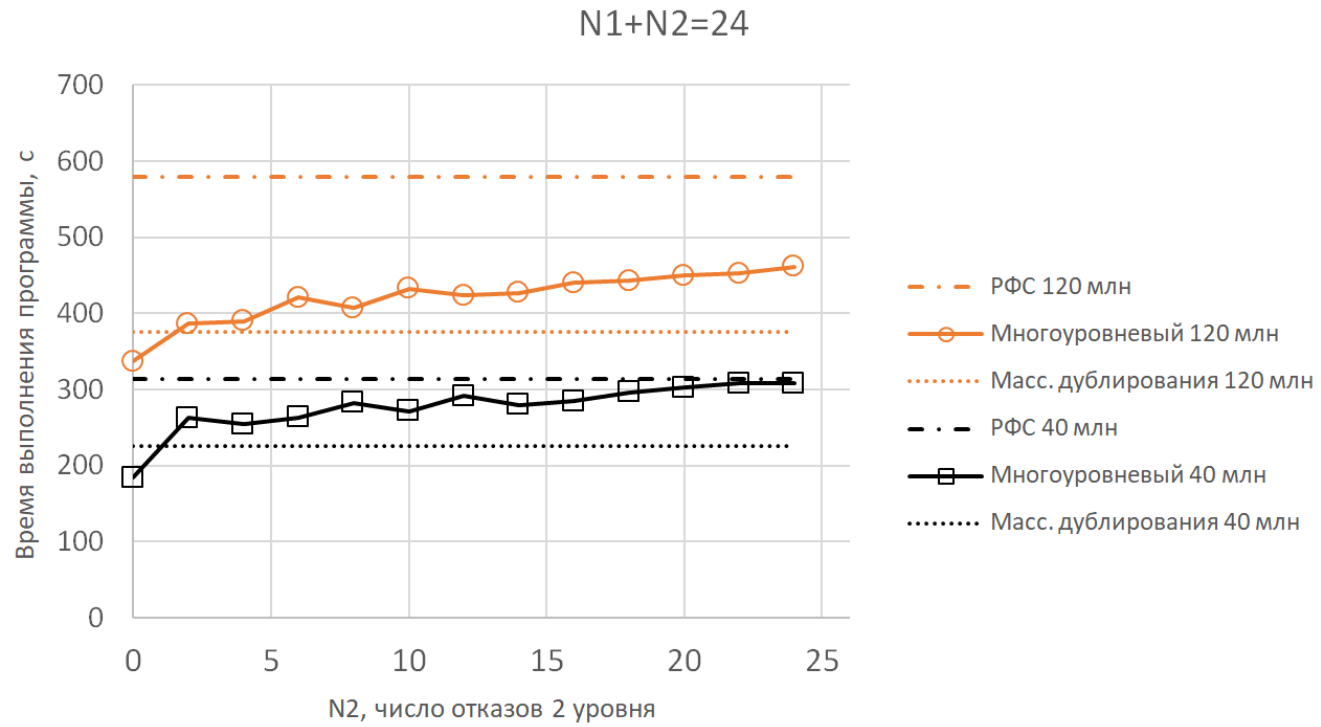

Рис.3 Время выполнения трех параллельных программ на сетках 40 и 120 млн. узлов при наличии отказов разных уровней, суммарное число которых равно 24.

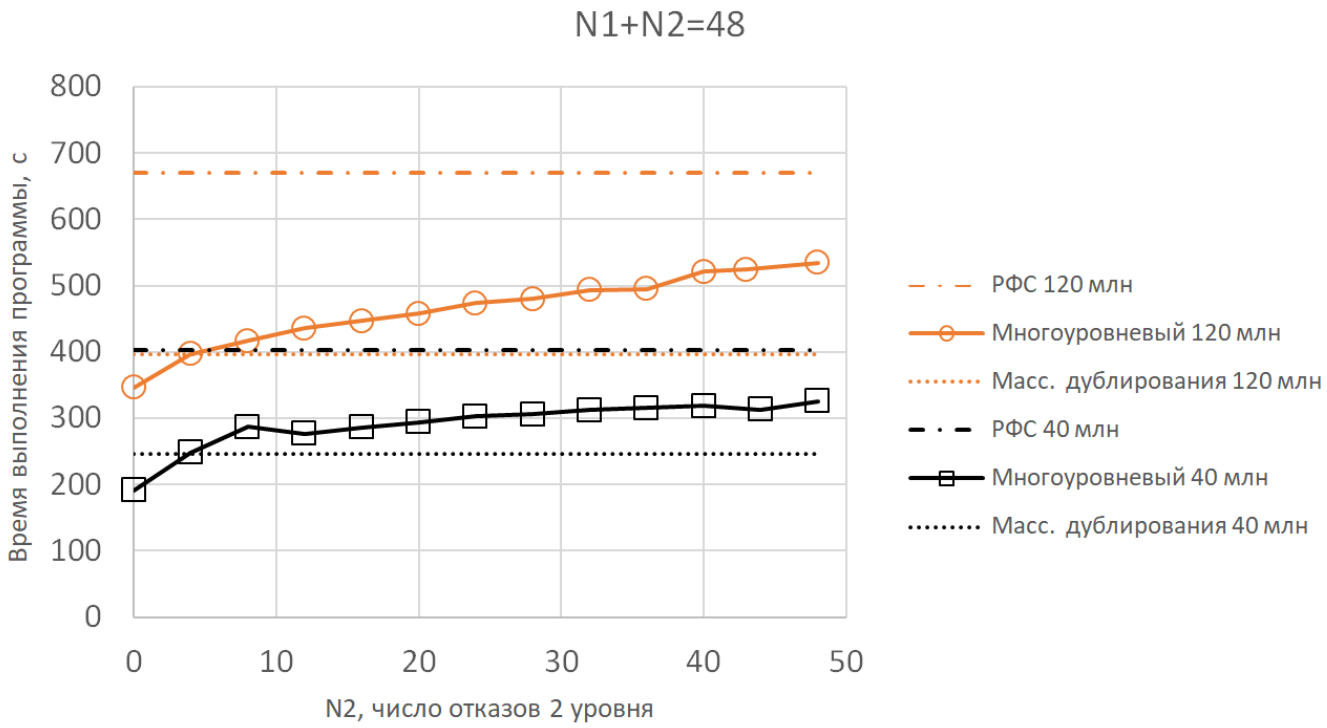

Рис.4 Время выполнения трех параллельных программ на сетках 40 и 120 млн. узлов при наличии отказов разных уровней, суммарное число которых равно 48.

Черный цвет соответствует расчетам для сетки, состоящей из 40 млн. узлов, оранжевый (более светлый) – 120 млн. узлов. Штрих-пунктиром отмечено время на работу программы с сохранением контрольных точек в распределенную файловую систему, ломаной линией отмечено время работы программы при использовании многоуровневого протокола сохранения контрольных точек, точками (пунктиром) отмечено время работы программы при массовом дублировании данных в оперативную память.

Для упрощения представления результатов было принято для программ с сохранением контрольных точек в распределенную файловую систему и в оперативную память (методом массового дублирования) осуществлять единичные отказы, то есть отказы только первого уровня ($n_1 \in \{12,24,48\}$, $n_2 = 0$). И даже в этом случае, накладные расходы на сохранение в распределенную файловую систему существенно больше по сравнению с другими методами сохранения. Отметим, что многоуровневый метод нуждается в информации о характеристиках отказов в системе (в рассматриваемых примерах – среднее время между единичными и двойными отказами). Как следствие наличие этой информации приводит к тому, что многоуровневый метод может иметь меньше накладных расходов, чем метод массового дублирования, так как последний должен иметь достаточно копий контрольных точек для восстановления в случае единичных и двойных отказов.

4. Заключение

При организации отказоустойчивых расчетов с помощью координированного сохранения контрольных точек, даже при наличии достаточного места в оперативной памяти для массового дублирования, важное значение приобретает информация о кратности отказов. Если предполагается, что происходят редкие кратные отказы, то эксперимент показал, что лучше использовать многоуровневый метод сохранения контрольных точек, а в случае частого возникновения кратных отказов, при возможности для сохранения контрольных точек лучше использовать метод массового дублирования в локальные устройства хранения. Сохранять контрольные точки только в распределенную файловую систему для отказоустойчивых расчетов даже на уровне пользователя не рекомендуется.

Литература

1. Elnozahy E. N. M., Alvisi L., Wang Y.-M., Johnson D. B. A survey of rollback-recovery protocols in message-passing systems. ACM Comput. Surv. 2002. Vol. 34, No 3, P. 375–408.
2. COMPUTATIONAL RESEARCH [Электронный ресурс] Режим доступа:http://crd.lbl.gov/departments/computer-science/CLaSS/research/BLCR/ (дата обращения: 1.04.2017).
3. Kogge P.M. et al. ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems — Tech. Report TR-2008-13. — Univ. of Notre Dame, CSE Dept. — 2008. / URL: http://www.cse.nd.edu/Reports/2008/TR-2008-13.pdf (дата обращения: 1.04.2017).
4. Fault Tolerance Research Hub [Электронный ресурс] Режим доступа: http://fault-tolerance.org/ (дата обращения: 1.04.2017).
5. Di S., Bouguerra M.S.; Bautista-Gomez L., Cappello F.; ,Optimization of multi-level checkpoint model for large scale HPC applications // Parallel and Distributed Processing Symposium, 2014 IEEE 28th International P. 1181-1190, 2014.
6. Scientific Cluster of Keldysh Institute of Applied Mathematics RAS. URL: http://imm6.keldysh.ru/~informer/ (дата обращения: 1.04.2017).
7. Бондаренко, А.А. Якобовский М.В. Обеспечение отказоустойчивости высокопроизводительных вычислений с помощью локальных контрольных точек // Вестник Южно-Уральского государственного университета. Серия «Вычислительная математика и информатика». 2014. Том. 3, No. 3. С. 20–36.
8. Cappello F., Geist A., Gropp W., Kale S., Kramer B., Snir M., Toward exascale resilience: 2014 update, Supercomputing Frontiers and Innovations, vol. 1, no. 1, pp. 5–28, 2014.

Fault Tolerance Mechanisms based on coordinated checkpointing

A.A. Bondarenko[1], P.A. Lyakhov[2], M.V. Yakobovskiy[1]

Keldysh Institute of Applied Mathematics of Russian Academy of Sciences[1], Moscow Institute of Physics and Technology (State University)[2]

In the world of high-performance computing, fault tolerance and application resilience are becoming some of the primary concerns because of increasing hardware failures. The current proposal for MPI fault tolerant is centered around the User-Level Failure Mitigation (ULFM) concept, which provides means for fault detection and recovery of the MPI layer. This approach does not address application-level recovery, which is currently left to application developers. In this work, we consider methods for coordinating checkpointing at the user-level: saving to a distributed file system, a multilevel checkpointing, a method of mass replication to local storage devices. An assessment is made of the applicability of these methods to the organization of fault-tolerant computations.

Keywords: parallel programming, MPI, ULFM, multi-level checkpoint, fault-tolerant computation.

References

1. Elnozahy E. N. M., Alvisi L., Wang Y.-M., Johnson D. B. A survey of rollback-recovery protocols in message-passing systems. ACM Comput. Surv. 2002. Vol. 34, No 3, P. 375–408.
2. COMPUTATIONAL RESEARCH URL:http://crd.lbl.gov/departments/computer-science/CLaSS/research/BLCR/ (accessed: 1.04.2017).
3. Kogge P.M. et al. ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems — Tech. Report TR-2008-13. — Univ. of Notre Dame, CSE Dept. — 2008. / URL: http://www.cse.nd.edu/Reports/2008/TR-2008-13.pdf (accessed: 1.04.2017).
4. Fault Tolerance Research Hub URL: http://fault-tolerance.org/ (accessed: 1.04.2017).
5. Di S., Bouguerra M.S.; Bautista-Gomez L., Cappello F.; ,Optimization of multi-level checkpoint model for large scale HPC applications // Parallel and Distributed Processing Symposium, 2014 IEEE 28th International P. 1181-1190, 2014.
6. Scientific Cluster of Keldysh Institute of Applied Mathematics RAS. URL: http://imm6.keldysh.ru/~informer/ (accessed: 1.04.2017).
7. Bondarenko A.A., Yakobovskiy M.V. Obespechenie otkazoustoychivosti vysokoproizvoditel'nykh vychisleniy s pomoshch'yu lokal'nykh kontrol'nykh tochek [Fault Toler- ance for HPC by Using Local Checkpoints]. Vestnik Yuzhno-Ural'skogo gosudarstven- nogo universiteta. Seriya «Vychislitel'naya matematika i informatika» [Bulletin of South Ural State University. Series: Computational Mathematics and Software Engineering]. 2014. Vol. 3, No. 3. P. 20–36. DOI: 10.14529/cmse140302.
8. Cappello F., Geist A., Gropp W., Kale S., Kramer B., Snir M., Toward exascale resilience: 2014 update, Supercomputing Frontiers and Innovations, vol. 1, no. 1, pp. 5–28, 2014.