УДК 519.6 DOI: 10.14529/cmse210105

МОДЕЛИРОВАНИЕ ВЛИЯНИЯ СИСТЕМЫ МОНИТОРИНГА ПРОИЗВОДИТЕЛЬНОСТИ НА ВЫПОЛНЕНИЕ КОЛЛЕКТИВНЫХ MPI ОПЕРАЦИЙ*

© 2021 А.А. Худолеева, К.С. Стефанов
Московский государственный университет имени М.В. Ломоносова
(119991 Москва, ул. Ленинские Горы, д. 1)
E-mail: khudoleeva.anna98@gmail.com, cstef@parallel.ru
Поступила в редакцию: 02.10.2020

Изучение параллельных программ с помощью средств мониторинга производительности — распространенная практика. Агент системы мониторинга для сбора данных о работе приложения периодически активируется во время счета этого приложения, внося помехи и занимая ресурсы. Однако вопрос об уровне влияния этих помех является слабо изученным, разработчики систем мониторинга зачастую не проводят исследования в этом направлении. В данной статье рассматриваются подходы к изучению влияния системы мониторинга производительности суперкомпьютера на пользовательские приложения. В качестве инструмента для измерения влияния агента системы мониторинга предлагается использовать коллективные MPI операции. Так, кроме обнаружения шума системы мониторинга, можно исследовать влияние системы мониторинга на сильно синхронизированные приложения. Время выполнения коллективных MPI операций изучается в присутствии программного средства, моделирующего работу агента системы мониторинга производительности. Оценивается уровень шума, который каждая из рассматриваемых коллективных операций в выбранной конфигурации запуска способна зафиксировать. В работе приводятся данные запусков инструмента с коллективными MPI операциями All-to-All, All-Reduce, Barrier. Найдено, что хорошей стабильностью и чувствительностью обладают операции All-to-All и Barrier.

Ключевые слова: суперкомпьютер, мониторинг производительности, шум системы мониторинга, замедление параллельных задач, моделирование влияния системы мониторинга.

ОБРАЗЕЦ ЦИТИРОВАНИЯ

Худолеева А.А., Стефанов К.С. Моделирование влияния системы мониторинга производительности на выполнение коллективных MPI операций // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2021. Т. 10, № 1. С. 62–74. DOI: 10.14529/cmse210105.

Введение

Современные суперкомпьютеры — это очень сложные, большие вычислительные комплексы, состоящие из множества компонентов. Рост возможностей системы, под которую пишется приложение, затрудняет его разработку и усложняет структуру этого приложения. Это ведет к повышению сложности оптимизации приложения под целевую платформу. Достижение максимальной производительности и наиболее полное использование предоставленных ресурсов — важная задача. Невозможность эффективно использовать средства суперкомпьютера ведет к замедлению работы программ и, следовательно, к задержке проведения научного исследования. Для решения данной проблемы оптимизации приложения можно использовать систему мониторинга производительности суперкомпьютера.

При проектировании средства мониторинга разработчики преследуют цель создания масштабируемого, эффективного инструмента, который будет предоставлять для анализа

*Статья рекомендована к публикации программным комитетом Международной конференции «Суперкомпьютерные дни в России – 2020».

необходимую информацию об используемых пользовательским процессом ресурсах. Данные могут характеризовать, например, взаимодействие приложения с различными частями системы: с процессором, сетью, памятью. Агент системы мониторинга запускается совместно с исследуемым приложением на тех же вычислительных узлах и с определенным периодом выполняет сбор данных и их отправку для дальнейшей обработки. Так как средство мониторинга разделяет ресурсы с приложением, оно вносит помехи в работу исследуемого приложения и негативно влияет на производительность. К данному моменту разработано достаточное количество различных систем мониторинга. Однако авторы этих систем часто утверждают, что влияние средств мониторинга незначительно, не приводя описания методов, позволяющих это влияние измерить.

Вопросу обнаружения влияния системы мониторинга производительности суперкомпьютера с помощью использования коллективных MPI операций и установления уровня этого влияния указанным методом посвящена эта работа.

Статья организована следующим образом. Раздел 1 посвящен описанию исследований влияния некоторых существующих систем мониторинга производительности и шума ОС на пользовательские задачи. В разделе 2 описываются выбранный инструмент для обнаружения шума, эксперименты, проведенные на СК, и результаты этих экспериментов. В заключении приводится краткая сводка результатов, полученных в работе, и указаны направления дальнейших исследований.

1. Обзор работ

За последние 20 лет было уделено много внимания созданию различных средств мониторинга производительности: Supermon [9], NWPerf [6], HPCToolkit [1], Performance Co-Pilot [13], LIKWID [9, 11], LDMS [2]. Авторы статей об инструментах HPCToolkit, Performance Co-Pilot, LIKWID говорят, что эффект, оказываемый этими средствами мониторинга на пользовательские приложения, является незначительным, однако не приводят результатов исследований, связанных с этим вопросом. Только для двух систем из списка [2, 6] проводилось исследование их влияния на пользовательские задачи.

Агентом системы мониторинга NWPerf [6] является модуль ядра Linux. Стандартный интервал сбора данных для этого средства — одна минута. Для этого инструмента было проведено исследование его влияния на коллективные операции All-to-All и All-Reduce. На 128 узлах, на 256 вычислительных ядрах запускался цикл из 10 000 коллективных операций с и без средства мониторинга. При частоте сбора данных (1 раз в 6 секунд), в десять раз превышающей стандартный режим, время выполнения операции All-to-All увеличилось на 27%, а операции All-Reduce — на 9,46%. Но при стандартной частоте прочие помехи в системе перекрывали шум от NWPerf, что свидетельствует о приемлемости частоты работы агента раз в минуту.

Достаточно подробно было исследовано влияние средства мониторинга LDMS [2] на работу различных компонент высокопроизводительной системы. Средой проведения экспериментов выступали две большие вычислительные системы. Тестирование с использованием бенчмарков показало, что инструмент LDMS при частоте работы 1 раз в секунду имеет незначительное влияние на производительность MPI коммуникаций.

Разница в полученных результатах в вопросе влияния системы мониторинга на MPI коммуникации значительная. Она может быть вызвана аппаратными и сетевыми отличиями суперкомпьютеров, на которых запускались системы мониторинга. Исследование системы LDMS [2] является более актуальным, так как проводилось на 10 лет позже исследования системы NWPerf [6]. Тем не менее, полученные в работах [2, 6] результаты

сложно обобщить, и вопрос влияния помех, вносимых системами мониторинга на производительность параллельных программ, остается открытым.

Хорошо исследованной является область обнаружения уровня влияния шума операционной системы на работу параллельных программ. Методы, применяемые в этой области, можно распространить на изучение шума системы мониторинга. В работе [3] при использовании большого количества процессов удалось установить, что несинхронизированный шум от операционной системы, возникающий периодически, является причиной значительного замедления коммуникаций Barrier и All-Reduce. В исследовании [7], посвященном оптимизации приложения SAGE на суперкомпьютере ASCI Q, было показано, что источником замедления работы приложения была низкая производительность операции All-Reduce. Авторы исследования [7] сделали вывод о том, что на некоторые хорошо синхронизированные приложения небольшой, но частый синхронный шум ОС оказывает сильное влияние.

Можно предположить, что коллективные операции являются чувствительными даже к малому шуму, создаваемому агентом системы мониторинга производительности, и их можно использовать для обнаружения помех, вносимых системой мониторинга в работу пользовательской задачи.

2. Методология исследования

Одним из самых популярных средств при программировании приложений для вычислительных кластеров является MPI — интерфейс, поддерживающий работу параллельных процессов с помощью передачи сообщений между ними. Из функционала, который доступен при использовании MPI, широко используются коллективные операции. В исследовании [5] проводился анализ 110 параллельных приложений с открытым исходным кодом. Минимальный функционал, который используют все рассмотренные в работе приложения — вызов коллективных операций.

Приложение может выполняться несимметрично, то есть в зависимости от ранга процессам назначаются роли, реализованные разными фрагментами кода. На большом вычислительном кластере нельзя гарантировать бесперебойность работы каждого узла, симметрию выполнения приложения на каждом ядре. Поэтому даже в приложениях, где процессы выполняют один и тот же фрагмент кода, стартовав синхронно, время выполнения последовательной части программы будет разным для разных процессов. Так как при запуске коллективной операции все процессы должны одновременно выполнить пересылку сообщений или синхронизацию, часть процессов будет простаивать до начала коммуникации, ожидая готовности к выполнению операции «опаздывающих» процессов. Поэтому время работы параллельного приложение измеряется по самому медленному процессу.

В силу тех фактов, что использование коллективных MPI операций является распространенной практикой и они могут стать весомым источником замедления в работе хорошо синхронизированной программы, важным является вопрос рассмотрения влияния агента системы мониторинга на выполнение коллективных операций. Задачей данной работы является исследование влияния системы мониторинга производительности на коллективные операции при использовании небольшого числа узлов суперкомпьютера. Возможность использования коллективных MPI операций, как средства для обнаружения шума, создаваемого системой мониторинга, и его уровня, исследуется ниже.

2.1. Программное средство, детектирующее шум

На рис. 1 представлена схема реализации программного средства — детектора шума — с помощью которого предлагается исследовать влияние шума системы мониторинга производительности. Измеряемой величиной при работе детектора шума является время выполнения цикла из *count* числа коллективных MPI операций. Рассматриваемыми коллективными MPI операциями в рамках данной работы являются: Barrier, All-Reduce, All-to-All.

```
start timer
do over iteration count
     MPI collective operation
stop timer
```

Рис. 1. Схема реализации инструмента для обнаружения шума

Для разработанного программного средства существует несколько свободных параметров, выбрав которые, можно определить конфигурацию запуска детектора шума. Конфигурацию инструмента будем считать подходящей, если при запуске в стандартных условиях, в отсутствии искусственно внедренного шума, эксперимент является воспроизводимым, то есть от запуска к запуску время выполнения коллективных операций изменяется в пределах допустимой погрешности. Также важной является способность инструмента фиксировать присутствие в системе дополнительного шума, чем выше чувствительность, тем лучше.

Свободные параметры включают в себя:

– выбор коллективной операции;
– количество узлов, на которых запущено программное средство, количество процессов;
– количество коллективных операций;
– для операций All-Reduce и All-toAll длина передаваемого сообщения.

2.2. Программное средство, моделирующее шум агента системы мониторинга

Было решено начать исследование возможности детектора обнаружить шум системы мониторинга с изучения времени работы детектора совместно с искусственным шумом. На рис. 2 представлен фрагмент кода источника шума — программы, которая моделирует влияние агента системы мониторинга производительности на работу пользовательского приложения.

В основе работы этого средства лежит цикл, в котором вычисляется сумма (значение суммы хранится в переменной *sum*) массива **a* объемом 250 000 слов (значение константы *SIZE*) типа long long. В *WORK_TIME* хранится доля секунды, которую работает нагрузка — суммирование массива. В параметр *tw* процедуры *nanosleep()* записана доля секунды, в течение которой сумма массива не вычисляется. Это значение дополняет *WORK_TIME* до 1 секунды. *BREAK_TIME* — общее время работы источника шума. Уровень шума в процентах — значение 100x*WORK_TIME*. Так, если речь идет об искусственном шуме 1%, значит каждую секунду цикл суммирования массива работает в течение 0,01 с и оставшуюся часть секунды бездействует. Стоит заметить, что величина искусственного шума 1%, создаваемого источником шума, оценивается на порядок выше, чем величина шума от агента реальной системы мониторинга.

```
start_time = MPI_Wtime();
do
{
     t_1 = MPI_Wtime();
     do
     {
        for (int i = 0; i < SIZE; ++i)
          sum += a[i];
        t_2 = MPI_Wtime();
     }
     while ((t_2 - t_1) < WORK_TIME);
     nanosleep(&tw, &tc);
     t_2 = MPI_Wtime();
}
while ((t_2 - start_time) < BREAK_TIME);
```

Рис. 2. Фрагмент реализации программного средства, моделирующего влияние агента системы мониторинга

2.3. Описание проведенных экспериментов

Исследование разработанных программных средств было выполнено на суперкомпьютере Ломоносов-2 [12]. Запуски проводились в тестовом разделе суперкомпьютера. Так удалось избежать ожидания в загруженной очереди основного раздела. На каждом узле установлен процессор Intel Haswell-EP E5-2697v3, имеющий 14 ядер. Лимит времени выполнения заданий на тестовом разделе составляет 15 минут. Этим фактом обусловлен выбор числа итераций в цикле для детектора шума. При запуске на суперкомпьютере использовалась библиотека OpenMPI [14].

Исследование детектора шума проходило в два этапа. Сначала необходимо было оценить стабильность работы разных вариантов детектора шума. В случае сильного разброса времени выполнения коллективных операций одного из вариантов детектора на системе Ломоносов-2, нужно отказаться от такой конфигурации, так как она не подходит для многократного использования в качестве инструмента, предназначенного для обнаружения шума. Причиной этого является отсутствие возможности получить воспроизводимое значение времени работы детектора без шума и, следовательно, отличить его от времени работы детектора в присутствии сторонней нагрузки. Далее, выбранные варианты детектора шума запускались совместно с источником шума. Полученная разница во времени выполнения коллективных операций в этих двух случаях сравнивалась с помощью статистических критериев, описанных в работе [4], и определялись чувствительные к дополнительной нагрузке конфигурации детектора шума.

В силу большого разнообразия возможностей проведения запусков была выбрана следующая конфигурация: программные средства запускались на 4 узлах суперкомпьютера Ломоносов-2, по 2 процесса на ядро (один процесс на виртуальное ядро, Hyper Threading включен). Для программы с операцией All-to-All также проводились тесты на 8 узлах. Модель агента системы мониторинга запускалась совместно с детектором на тех же узлах, но на одном ядре одного узла.

2.4. Результаты измерений

На рисунках ниже и в табл. 1, 2 представлены результаты измерений для каждой конфигурации детектора.

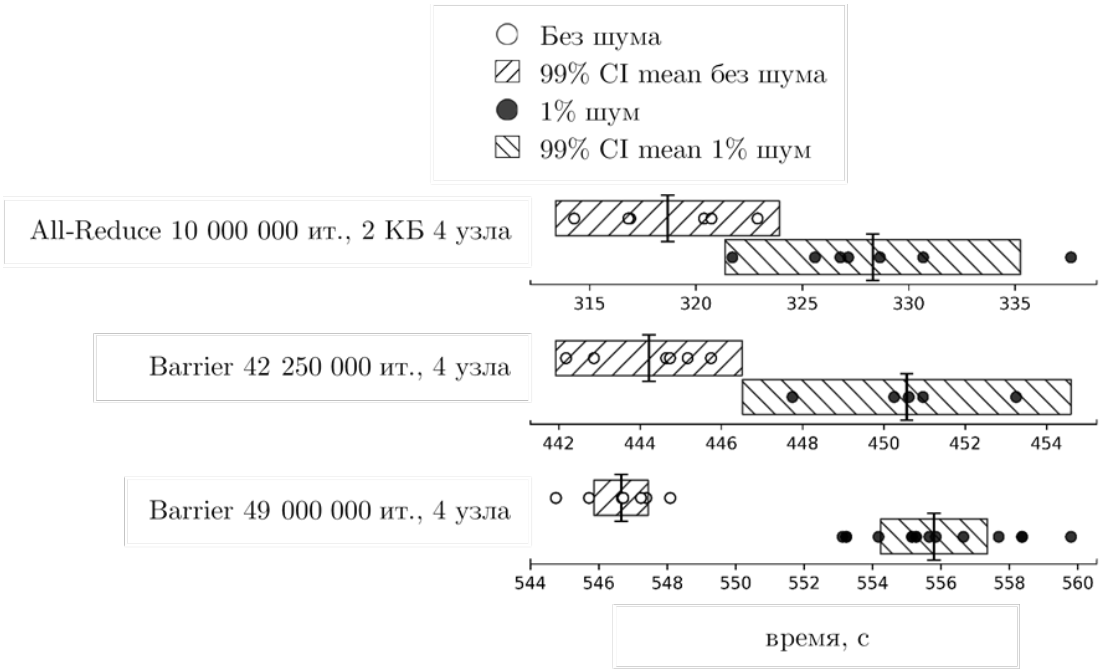

Рис. 3. Распределение времени для операций All-Reduce, Barrier

Таблица 1

Результаты запусков для I: All-Reduce, 10 000 000 ит., 2 КБ, 4 узла;
II: Barrier, 42 250 000 ит., 4 узла; III: Barrier, 49 000 000 итераций, 4 узла

		I	II	III
Запуски без шума	Среднее значение, с	318,7	444,2	546,66
	Стандартное отклонение, с	3,19	1,39	1,03
	Количество запусков	6	6	8
Запуски с шумом 1%	Среднее значение, с	328,3	450,6	555,90
	Стандартное отклонение, с	4,95	1,96	2,09
	Количество запусков	7	5	11

All-Reduce, 1 000 000 операций (рис. 3, табл. 1) демонстрирует разброс во времени выполнения детектора без шума. Также конфигурация не обнаруживает шум 1%–99% CI mean времени работы детектора без шума и с шумом 1% пересекаются. Данная конфигурация дальше не используется. Для детектора с 42 250 000 итераций (рис. 3, табл. 1) операции Barrier доверительные интервалы касаются. Было решено проверить эту конфигурацию детектора при большем числе операций Barrier. Так, для операции Barrier, 49 000 000 итераций, 4 узла на рис. 3 видно, что значения времени работы детектора с шумом 1% превышают доверительный интервал времени работы детектора без шума. Эта конфигурация детектора обнаруживает искусственный шум 1% и время работы этого детектора воспроизводимо.

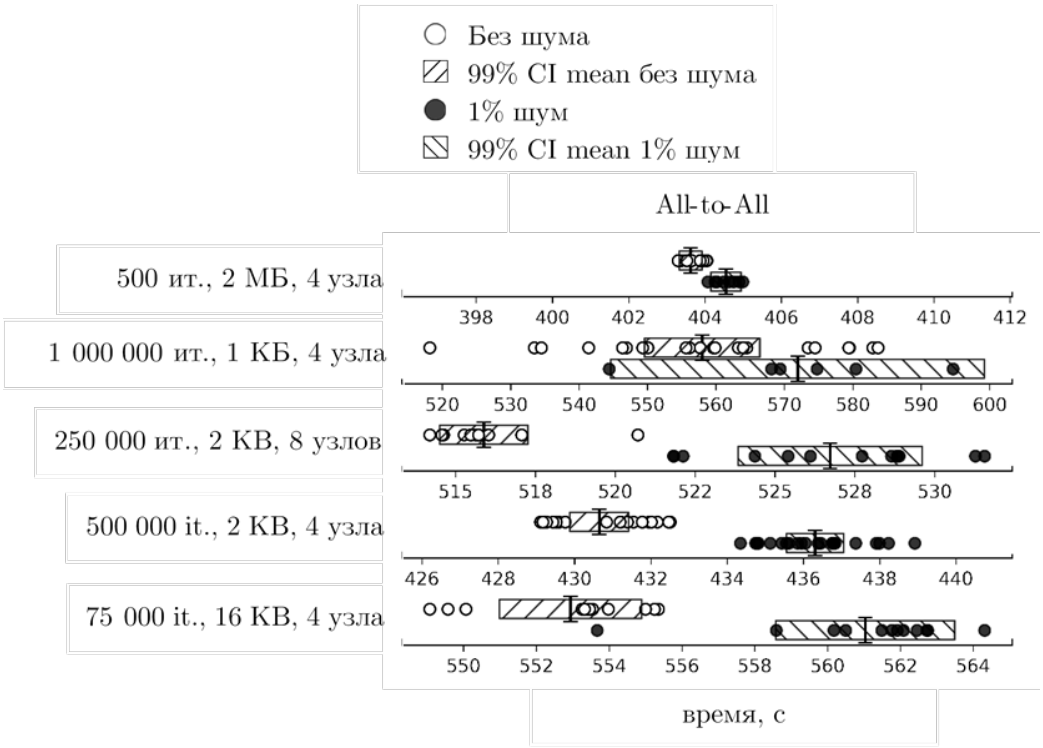

Рис. 4. Распределение времени для операции All-to-All

Таблица 2

Результаты запусков для I: All-to-All 500 ит., 2 МБ, 4 узла;
II: All-to-All 1 000 000 ит., 1 КБ, 4 узла; III: All-to-All 250 000 ит., 2 КБ, 8 узлов;
IV: All-to-All 500 000 ит., 2 КБ, 4 узла; V: All-to-All 75 000 ит., 16 КБ, 4 узла

		I	II	III	IV	IV
Чистые запуски	Среднее значение, с	403,6	558,0	515,9	430,7	552,9
	Стандартное отклонение, с	0,3	14,3	1,6	1,3	2,2
	Количество запусков	10	23	13	22	12
Запуски с шумом 1%	Среднее значение, с	404,5	572,0	526,7	436,3	561,0
	Стандартное отклонение, с	0,3	16,6	3,4	1,2	2,7
	Количество запусков	8	6	13	22	12

Для All-to-All, 500 итераций, 2 МБ, 4 узла на рис. 4 видно, что время работы детектора с шумом 1% отличается меньше чем на 1 секунду от времени работы детектора без шума. Эта конфигурация детектора не чувствительна к шуму 1%. В случае All-to-All, 1 000 000 итераций, 1 КВ, 4 узла время «чистых» запусков имеет сильный разброс. При таких параметрах детектора уловить разницу между «чистыми» запусками и запусками с шумом 1% нельзя. Причины плохой воспроизводимости времени работы детектора в этой конфигурации неизвестны. All-to-All, 75 000 операций, 16 КБ, 4 узла (рис. 4, табл. 2) можно считать пригодным инструментом для обнаружения шума. Детекторы All-to-All, 250 000 итераций, 8 узлов и All-to-All, 500 000 итераций, 4 узла (рис. 4, табл. 2) с размером сообщения 2 КБ показывают наиболее стабильный результат, с хорошей возможностью обнаружения шума 1%.

Во время запусков на суперкомпьютере выбросы появлялись для всех выбранных коллективных операций. Возможно, на появление разницы времени запусков влияют конкретные узлы, которые выделяются на суперкомпьютере, задержки во взаимодействии между ними. Причины появления выбросов подробно не исследовались.

2.5. Тестирование чувствительности

Выше были приведены результаты, демонстрирующие способность разработанных программных средств улавливать искусственный шум 1%, создаваемый моделью агента системы мониторинга производительности. Ниже на графиках приведены результаты экспериментов, в которых уровень шума постепенно снижался до порога чувствительности (до верхней границы доверительного интервала среднего значения времени работы детектора без шума). Для тестирования были выбраны детекторы, с помощью которых можно обнаружить шум 1%:

– All-to-All 500 000 итераций, 2 КБ, 4 узла (табл. 3);
– All-to-All 250 000 итераций, 2 КБ, 8 узла (табл. 4);
– All-to-All 75 000 итераций, 16 КБ, 4 узла (табл. 5);
– Barrier 49 000 000 итераций, 4 узла (табл. 6).

На рис. 5 и в табл. 3–6 при каждом указанном уровне нагрузки представлено среднее арифметическое времени работы детектора шума по всем соответствующим запускам. По графикам на рис. 5 видно, что при нагрузке в 0,5% шума для детекторов, использующих операции Barrier и All-to-All, время с шумом и без можно различить с хорошей надежностью. При уровне шума 0,3% для детектора, основанного на операции All-to-All, 2 КБ наблюдается достижение границы чувствительности. При дальнейшем понижении уровня шума доверительные интервалы среднего значения времени работы детектора с шумом и без начинают пересекаться. Детекторы All-to-All, 16 КБ и Barrier, 49 000 000 итераций также обнаруживает шум до 0,3%. Причем на время работы операции Barrier шум 0,01% и шум 0,3% влияет одинаково. Чувствительность операции Barrier к столь малому шуму 0,01% требует более тщательной проверки — проведения повторных экспериментов — так как ни для какого другого варианта детектора шума такая чувствительность не была замечена. Видно, что все рассмотренные конфигурации детектора шума обнаруживают малый шум до 0,3%.

Таблица 3

Результаты тестирования чувствительности All-to-All 500 000 итераций, 2 КБ, 4 узла

Уровень шума	0%	0,01%	0,05%	0,1%	0,3%	0,5%	0,7%	1%
Среднее значение, с	431	432	432	433	434	435	436	436
Стандартное отклонение, с	1,3	1,1	2,1	1,5	1,7	1,5	2,2	1,2
Количество запусков	22	10	4	9	6	7	9	22

Таблица 4

Результаты тестирования чувствительности All-to-All 250 000 итераций, 2 КБ, 8 узлов

Уровень шума	0%	0,01%	0,05%	0,1%	0,3%	0,5%	0,7%	1%
Среднее значение, с	516	517	523	519	519	526	529	527
Стандартное отклонение, с	1,6	3,4	0,7	4,3	1,1	0,9	2,7	3,4
Количество запусков	13	5	5	5	5	5	4	13

Таблица 5

Результаты тестирования чувствительности Barrier, 49 000 000 итераций, 4 узла

Уровень шума	0%	0,01%	0,05%	0,1%	0,3%	0,5%	0,7%	1%
Среднее значение, с	547	552	549	551	552	553	555	556
Стандартное отклонение, с	1,0	1,5	1,3	1,0	1,7	0,7	0,9	2,1
Количество запусков	8	5	5	5	5	5	5	11

Таблица 6

Результаты тестирования чувствительности All-to-All 75 000 ит., 16 КБ, 4 узла

Уровень шума	0%	0,01%	0,05%	0,1%	0,3%	0,5%	0,7%	1%
Среднее значение, с	553	555	556	557	560	560	561	561
Стандартное отклонение, с	2,2	0,2	0,3	1,2	1,9	0,3	0,7	2,7
Количество запусков	12	5	5	5	5	5	5	12

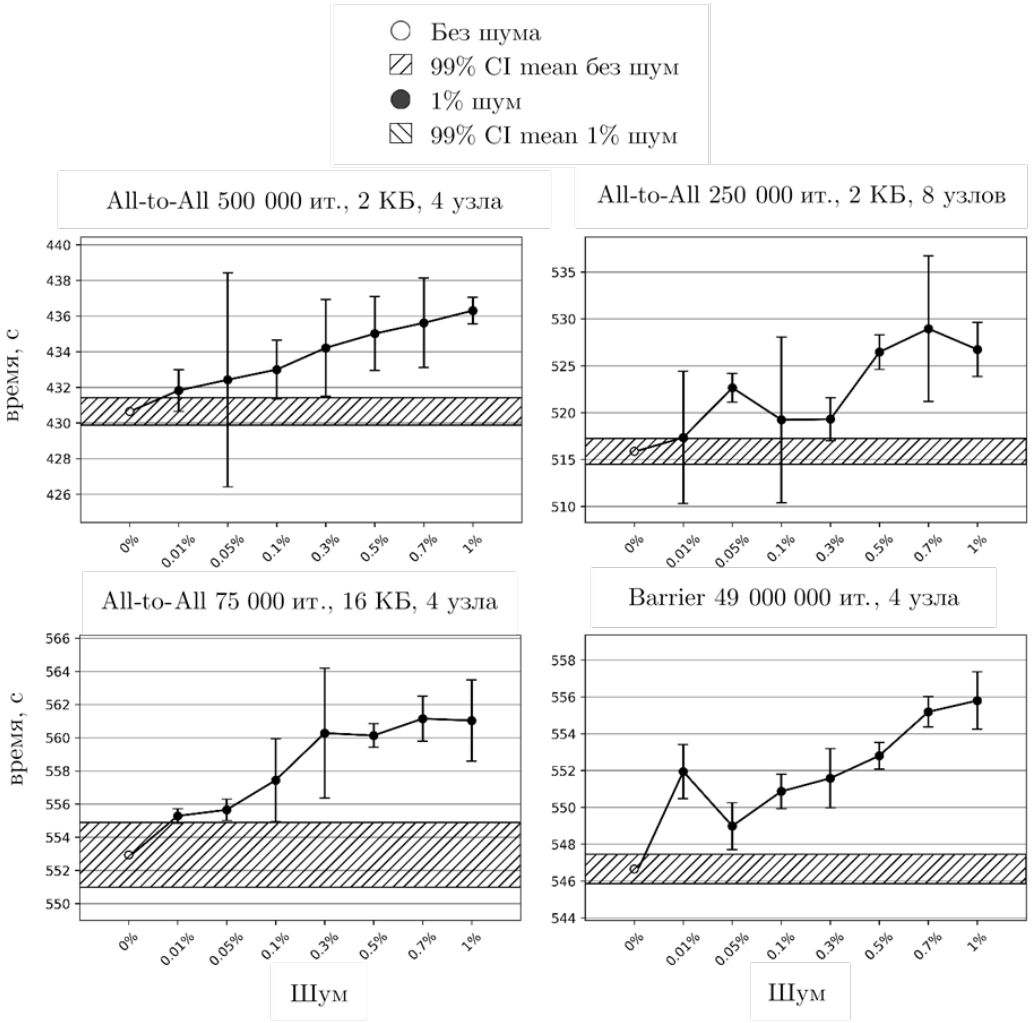

Рис. 5. Тестирование чувствительности

Заключение

Были разработаны два программных средства — источник шума, предназначенный для моделирования влияния работы агента системы мониторинга производительности, и детектор шума. В качестве детектора шума были опробованы программы с различными коллективными MPI операциями. В работе рассматривалось замедление времени выполнения детектора с шумом относительно времени выполнения детектора без дополнительной нагрузки. Так, если замедление работы коллективных MPI операций статистически значимо, то коллективная операция пригодна для обнаружения дополнительной малой нагрузки в системе.

Для операции All-to-All проводились запуски с различной длиной сообщения. Оказалось, что наиболее стабильным из рассмотренных является детектор с длиной пересылаемого сообщения 2 КБ, протестированный на 4-х и 8-ми узлах суперкомпьютера.

Детектор, использующий операцию Barrier, при большом числе итераций (время выполнения от 7 минут) также является чувствительным к искусственному шуму 1%. При использовании All-Reduce стабильная конфигурация для суперкомпьютера не была найдена.

Для операции All-to-All была обнаружена граница чувствительности к нагрузке, которую создает источник шума. Это значение составляет 0,3% для операции All-to-All. Операция Barrier также чувствительна к шуму 0,3%. Шум 0,3% является малым, поэтому можно сказать, что операции All-to-All и Barrier обладают хорошей чувствительностью. Эти коллективные операции являются подходящими для обнаружения влияния малой дополнительной нагрузки, и их можно использовать для изучения шума от системы мониторинга производительности суперкомпьютера.

В дальнейшей работе планируется исследовать чувствительность операций All-to-All и Barrier к шуму реальной системы мониторинга производительности суперкомпьютера Ломоносов-2 [10]. Будет исследовано время работы различных конфигураций детектора шума и отобраны те из них, которые являются чувствительными к шуму агента системы мониторинга, работающего в стандартном режиме. С помощью отобранных конфигураций детектора планируется сравнить уровень влияния разных режимов работы агента системы мониторинга производительности на синхронизированные приложения.

Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 19-07-00940.

Работа выполнена с использованием оборудования Центра коллективного пользования сверхвысокопроизводительными вычислительными ресурсами МГУ имени М.В. Ломоносова.

Литература

1. Adhianto L., Banerjee S., Fagan M., et al. HPCTOOLKIT: tools for performance analysis of optimized parallel programs // Concurr. Comput. Pract. Exp. 2009. Vol. 22, no. 6. P. 685–701. DOI: 10.1002/cpe.1553.
2. Agelastos A., Allan B., Brandt J., et al. The Lightweight Distributed Metric Service: A Scalable Infrastructure for Continuous Monitoring of Large Scale Computing Systems and Applications // International Conference for High Performance Computing, Networking, Storage and Analysis, SC14 (New Orleans, LA, USA, Nov. 2014). IEEE, 2014. P. 154–165. DOI: 10.1109/SC.2014.18.
3. Beckman P., Iskra K., Yoshii K., et al. Benchmarking the effects of operating system interference on extreme-scale parallel machines // Cluster Comput. 2008. Vol. 11, no. 1. P. 3–16. DOI: 10.1007/s10586-007-0047-2.
4. Hoefler T., Belli R. Scientific benchmarking of parallel computing systems // Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on, SC '15 (New York, New York, USA, Nov. 2015). ACM Press, 2015. P. 1–12. DOI: 10.1145/2807591.2807644.
5. Laguna I., Marshall R., Mohror K., et al. A large-scale study of MPI usage in open-source HPC applications // Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (New York, NY, USA, Nov. 2019). ACM, 2019. P. 1–14. DOI: 10.1145/3295500.3356176.
6. Mooney R., Schmidt K.P., Studham R.S. NWPerf: a system wide performance monitoring tool for large Linux clusters // 2004 IEEE International Conference on Cluster Computing. (San Diego, CA, USA, Sept. 2004). IEEE, 2004. P. 379–389. DOI: 10.1109/CLUSTR.2004.1392637.

7. Petrini F., Kerbyson D.J., Pakin S. The Case of the Missing Supercomputer Performance // Proceedings of the 2003 ACM/IEEE conference on Supercomputing, SC '03 (New York, New York, USA, Nov. 2003). ACM Press, 2003. P. 55. DOI: 10.1145/1048935.1050204.
8. Rohl T., Eitzinger J., Hager G., et al. LIKWID Monitoring Stack: A Flexible Framework Enabling Job Specific Performance monitoring for the masses // 2017 IEEE International Conference on Cluster Computing, CLUSTER (Honolulu, HI, USA, Sept. 2017). IEEE, 2017. P. 781–784. DOI: 10.1109/CLUSTER.2017.115.
9. Sottile M.J., Minnich R.G. Supermon: a high-speed cluster monitoring system // Proceedings. IEEE International Conference on Cluster Computing (Chicago, IL, USA, USA, Sept. 2002). IEEE Comput. Soc, 2002. P. 39–46. DOI: 10.1109/CLUSTR.2002.1137727.
10. Stefanov K., Voevodin V., Zhumatiy S., et al. Dynamically Reconfigurable Distributed Modular Monitoring System for Supercomputers (DiMMon) // Procedia Computer Science. Elsevier B.V., 2015. P. 625–634. DOI: 10.1016/j.procs.2015.11.071.
11. Treibig J., Hager G., Wellein G. LIKWID: A Lightweight Performance-Oriented Tool Suite for x86 Multicore Environments // 2010 39th International Conference on Parallel Processing Workshops (San Diego, CA, USA, Sept. 2010). IEEE, 2010. P. 207–216. DOI: 10.1109/ICPPW.2010.38.
12. Voevodin Vl.V., Antonov A.A., Nikitenko D.A., et al. Supercomputer Lomonosov-2: Large Scale, Deep Monitoring and Fine Analytics for the User Community // Supercomput. Front. Innov. 2019. Vol. 6, no. 2. P. 4–11. DOI: 10.14529/jsfi190201.
13. Performance Co-Pilot. URL: https://pcp.io/ (дата обращения: 27.09.2020)
14. Open MPI: Open Source High Performance Computing. URL: https://www.open-mpi.org/ (дата обращения: 27.09.2020)

Худолеева Анна Александровна, студентка, кафедра суперкомпьютеров и квантовой информатики, Московский государственный университет имени М.В. Ломоносова (Москва, Российская Федерация)

Стефанов Константин Сергеевич, к.ф.-м.н., старший научный сотрудник, научно-исследовательский вычислительный центр Московского государственного университета имени М.В. Ломоносова (Москва, Российская Федерация)

DOI: 10.14529/cmse210105

MODELING INFLUENCE OF MONITORING SYSTEM ON PERFORMANCE OF MPI COLLECTIVE OPERATIONS

© 2021 A.A. Khudoleeva, K.S. Stefanov
Lomonosov Moscow State University (GSP-1, Leninskie Gory 1, Moscow, 119991 Russia)
E-mail: khudoleeva.anna98@gmail.com, cstef@parallel.ru
Received: 02.10.2020

Studying parallel program with the means of monitoring systems is a common practice. To collect data about application, monitoring system agent activates periodically during the run of application, occupying resources and causing perturbation. Monitoring system developers often ignore studying the problem of monitoring tools interference into application performance, this problem remains poorly examined. This article discusses ways to study influence of supercomputer monitoring system on users' applications. We suggest to use MPI collective operations as a tool to measure this influence. This method also allows to estimate influence of monitoring system noise on a synchronized application. MPI collective operations are measured in presence of injected noise generated by the program that imitates interference of monitoring tool. We estimate the noise level that each of the used collective operations is capable to sense in chosen configuration. All-to-All, All-Reduce and Barrier are used in the noise detection tool. We find parameters for All-to-All and Barrier operations to perform stably and detect low noise level.

Keywords: supercomputer, performance monitoring, monitoring system noise, parallel job slowdown, modeling influence of monitoring system.

FOR CITATION

Khudoleeva A.A., Stefanov K.S. Modeling Influence of Monitoring System on Performance of MPI Collective Operations. *Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering.* 2021. Vol. 10, no. 1. P. 62–74. (in Russian) DOI: 10.14529/cmse210105.

This paper is distributed under the terms of the Creative Commons Attribution-Non Commercial 3.0 License which permits non-commercial use, reproduction and distribution of the work without further permission provided the original work is properly cites.

References

1. Adhianto L., Banerjee S., Fagan M., et al. HPCTOOLKIT: tools for performance analysis of optimized parallel programs. Concurr. Comput. Pract. Exp. 2009. Vol. 22, no. 6. P. 685–701. DOI: 10.1002/cpe.1553.
2. Agelastos A., Allan B., Brandt J., et al. The Lightweight Distributed Metric Service: A Scalable Infrastructure for Continuous Monitoring of Large Scale Computing Systems and Applications. International Conference for High Performance Computing, Networking, Storage and Analysis, SC14 (New Orleans, LA, USA, Nov. 2014). IEEE, 2014. P. 154–165. DOI: 10.1109/SC.2014.18.
3. Beckman P., Iskra K., Yoshii K., et al. Benchmarking the effects of operating system interference on extreme-scale parallel machines. Cluster Comput. 2008. Vol. 11, no. 1. P. 3–16. DOI: 10.1007/s10586-007-0047-2.
4. Hoefler T., Belli R. Scientific benchmarking of parallel computing systems. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on, SC '15 (New York, New York, USA, Nov. 2015). ACM Press, 2015. P. 1–12. DOI: 10.1145/2807591.2807644.
5. Laguna I., Marshall R., Mohror K., et al. A large-scale study of MPI usage in open-source HPC applications. Proceedings of the International Conference for High Performance

Computing, Networking, Storage and Analysis (New York, NY, USA, Nov. 2019). ACM, 2019. P. 1–14. DOI: 10.1145/3295500.3356176.

6. Mooney R., Schmidt K.P., Studham R.S. NWPerf: a system wide performance monitoring tool for large Linux clusters. 2004 IEEE International Conference on Cluster Computing (San Diego, CA, USA, Sept. 2004). IEEE, 2004. P. 379–389. DOI: 10.1109/CLUSTR.2004.1392637.
7. Petrini F., Kerbyson D.J., Pakin S. The Case of the Missing Supercomputer Performance. Proceedings of the 2003 ACM/IEEE conference on Supercomputing, SC '03 (New York, New York, USA, Nov. 2003). ACM Press, 2003. P. 55. DOI: 10.1145/1048935.1050204.
8. Rohl T., Eitzinger J., Hager G., et al. LIKWID Monitoring Stack: A Flexible Framework Enabling Job Specific Performance monitoring for the masses. 2017 IEEE International Conference on Cluster Computing, CLUSTER (Honolulu, HI, USA, Sept. 2017). IEEE, 2017. P. 781–784. DOI: 10.1109/CLUSTER.2017.115.
9. Sottile M.J., Minnich R.G. Supermon: a high-speed cluster monitoring system. Proceedings. IEEE International Conference on Cluster Computing (Chicago, IL, USA, USA, Sept. 2002). IEEE Comput. Soc, 2002. P. 39–46 DOI: 10.1109/CLUSTR.2002.1137727.
10. Stefanov K., Voevodin V., Zhumatiy S., et al. Dynamically Reconfigurable Distributed Modular Monitoring System for Supercomputers (DiMMon). Procedia Computer Science. Elsevier B.V., 2015. P. 625–634 DOI: 10.1016/j.procs.2015.11.071.
11. Treibig J., Hager G., Wellein G. LIKWID: A Lightweight Performance-Oriented Tool Suite for x86 Multicore Environments. 2010 39th International Conference on Parallel Processing Workshops (San Diego, CA, USA, Sept. 2010). IEEE, 2010. P. 207–216. DOI: 10.1109/ICPPW.2010.38.
12. Voevodin Vl.V., Antonov A.A., Nikitenko D.A., et al. Supercomputer Lomonosov-2: Large Scale, Deep Monitoring and Fine Analytics for the User Community. Supercomput. Front. Innov. 2019. Vol. 6, no. 2. P. 4–11. DOI: 10.14529/jsfi190201.
13. Performance Co-Pilot. URL: https://pcp.io/ (accessed: 27.09.2020).
14. Open MPI: Open Source High Performance Computing. URL: https://www.open-mpi.org/ (accessed: 27.09.2020).