7. Дерюгин Ю. Н., Козелков А. С., Спиридонов В. Ф., Циберев К. В., Шагалиев Р. М. Многофункциональный высокопараллельный пакет программ ЛОГОС для решения задач тепломассопереноса и прочности //. С.-Петербургский научный форум «Наука и общество»: сб. тез. С.-Пб: Изд-во Политех. ун-та, 2012.

8. Речкин В. Н., Спиридонов В. Ф., Циберев К. В. и др. Пакет программ ЛОГОС. Модуль решения квазистатических задач прочности и модального анализа // XIII Международный семинар «Супервычисления и математическое моделирование»: сб. докл.. – Саров: ФГУП «РФЯЦ-ВНИИЭФ», 2011.

9. Вентцель Е. С., Овчаров Л. А. Теория вероятностей. М.: Наука, 1973.

10. Chen M.-T., Ali A. An efficient and robust integration technique for applied random vibration analysis // Computers & Structures. – 1998. – V. 66, no. 6. –P. 785 – 798.

11. Harichandran R. Random vibration under propagating excitation: closed-form solutions // Journal of Engineering Mechanics. – 1992. – V. 118. – P. 575 – 586.

LOGOS SOFTWARE PACKAGE. COMPUTATIONAL METHOD FOR THE RESPONSE OF STRUCTURES UNDER BROAD-BAND RANDOM VIBRATION

A. Yu. Yeremenko, S. S. Kosarim, A. O. Naumov, V. N. Rechkin, A. I. Chembarov

Russian Federal Nuclear Center –
All-Russian Research Institute of Experimental Physics, Sarov

The paper describes a computational method for the response of a structure under the effect of the broad-band random vibration. We describe the computations of the response of the structure to the loading applied both to the fixed and not-fixed parts of the structure. We provide basic ratios to calculate the response of the structure as for travel, velocities and acceleration, and stresses. The realized methods of the response integration are described and compared. The solution of practical problems is shown.

Key words: LOGOS software package, broad-band random vibration, probabilistic load, structure response computation.

УДК 519.6

ПРОТОКОЛ ПАРАЛЛЕЛЬНОГО ВЫПОЛНЕНИЯ КОМАНД СИСТЕМНОГО АДМИНИСТРИРОВАНИЯ НА УЗЛАХ МНОГОПРОЦЕССОРНОГО ВЫЧИСЛИТЕЛЬНОГО КОМПЛЕКСА

Е. В. Ерёмин, Н. Н. Залялов

Российский федеральный ядерный центр –
Всероссийский НИИ экспериментальной физики, Саров

Предложен протокол и набор программ удалённого доступа к вычислительным узлам многопроцессорного вычислительного комплекса. Традиционные средства удалённого доступа, такие как SSH или PDSH выполняют запрос к n узлам последовательно, что имеет асимптотическую сложность $O(n)$. Сократить сложность до $O(\log n)$ можно, используя так называемое дерево запроса – способ разделить исходное множество узлов на части и работать

с ними одновременно Ответы группируются для сокращения объёма данных и более удобного анализа результатов. Реализованы внутренние обработчики некоторых команд операционной системы UNIX/Linux (cat, grep, ls, ps и т. д.), что позволяет избежать накладных расходов на порождение процессов, выделение памяти и т. д.

Ключевые слова: запрос, ответ, сообщение, утилита, клиент, сервер, удалённый доступ, удалённый вызов процедур.

Введение

Системное администрирование многопроцессорного вычислительного комплекса (МВК) основано на применении средств удалённого доступа к множеству вычислительных узлов. Под системным администрированием здесь понимается не столько предварительная настройка параметров системного программного обеспечения и оборудования вычислительного поля, сколько поиск исключительных, проблемных ситуаций «на лету» в процессе производственного счёта.

Предварительная настройка вычислительного поля и системного программного обеспечения имеет сравнительно слабые ограничения по длительности. Диагностика и устранение проблем требуют от средств удалённого доступа высокого быстродействия. Ожидание результата в течение нескольких десятков секунд от сотен и тысяч вычислительных узлов часто неприемлемо, поскольку ситуация за это время может измениться.

Одним из видов удалённого доступа является запрос на удалённое выполнение команды и получение результата. Для небольшого вычислительного комплекса можно использовать известные средства (SSH [1], PDSH [2]). Однако с ростом числа вычислительных узлов эффективность их применения уменьшается. При выполнении запроса на нескольких вычислительных узлах, длительность и объём результата растут пропорционально числу вычислительных узлов. Поскольку запрос выполняется на фоне производственного счёта, средства удалённого доступа должны иметь минимум накладных расходов. Традиционные средства порождают цепочку процессов для выполнения каждой команды.

В данной работе предлагается протокол, в котором сокращение длительности запроса достигается выполнением команды на нескольких узлах одновременно. Сокращение объёма данных достигается группировкой одинаковых результатов выполнения команд. Снижение накладных расходов достигается уменьшением числа порождаемых на вычислительном узле процессов для выполнения команды. Помимо прочего, к данному протоколу предъявляется требование универсальности, поскольку для выявления конкретных проблем существуют системы мониторинга, например [3].

Одним из способов выполнения команды на нескольких узлах одновременно является формирование дерева запроса. Дерево запроса как средство ускорения операций удалённого доступа к вычислительным узлам известно, и применяется, в частности, в системе управления пакетной обработки заданий Slurm [4]. Однако, в настоящей работе рассматривается не запуск параллельных задач, а оперативное выполнение команд системного администрирования.

1. Описание протокола

Протокол определяет процедуры взаимодействия программных компонент и формат передаваемых и принимаемых по сети сообщений. Программными компонентами являются утилита командной строки, корневой системный процесс, системные процессы вычислительных узлов. Пользователь взаимодействует с утилитой командной строки, выполняемой на инструментальном сервере. Утилита командной строки взаимодействует по сети с корневым системным процессом, который выполняется на управляющем сервере, а тот, в свою очередь – с системными процессами, выполняемыми в вычислительных узлах МВК. Корневой системный процесс отличается от

системных процессов вычислительных узлов только тем, что утилита командной строки обращается к нему по умолчанию. В остальном они идентичны.

Имеются следующие типы протокольных сообщений: прямой запрос, прямой ответ, групповой запрос, групповой ответ. Прямой запрос есть сообщение, передаваемое узлом-источником узлу-приёмнику, в котором узлу-приёмнику предписывается выполнить определённую команду. Прямой ответ есть ответное сообщение протокола, содержащее результат выполнения команды, которое отправляется узлом-приёмником узлу-источнику. Групповой запрос и групповой ответ подразумевают выполнение определённой команды на заданном множестве вычислительных узлов и получение ответов от них. С целью минимизации накладных расходов, в качестве транспортного механизма взаимодействия по сети используется UDP (Unreliable Datagram Protocol) [5].

Запрос и ответ – формируемые сообщения протокола. Сообщение протокола – последовательность байт, которая однозначно интерпретируется принимающей стороной. В процессе формирования сообщения, в него помещается команда, которую необходимо выполнить, и её параметры. В процессе интерпретации ответного сообщения из него извлекается результат выполнения команды.

1.1. Схема взаимодействия программных компонент

Традиционные средства удалённого доступа выполняют последовательные запросы к запрошенному множеству узлов (рис. 1,а). Если разделить исходное множество узлов на несколько частей и посылать запросы с нескольких серверов одновременно, то длительность выполнения операции сократится. В качестве серверов могут выступать узлы исходного множества. Части исходного множества можно, в свою очередь, разделить, увеличив, таким образом, степень параллельности выполнения запросов. Продолжая рекурсивно делить исходное множество узлов, получаем дерево запроса (рис. 1,б).

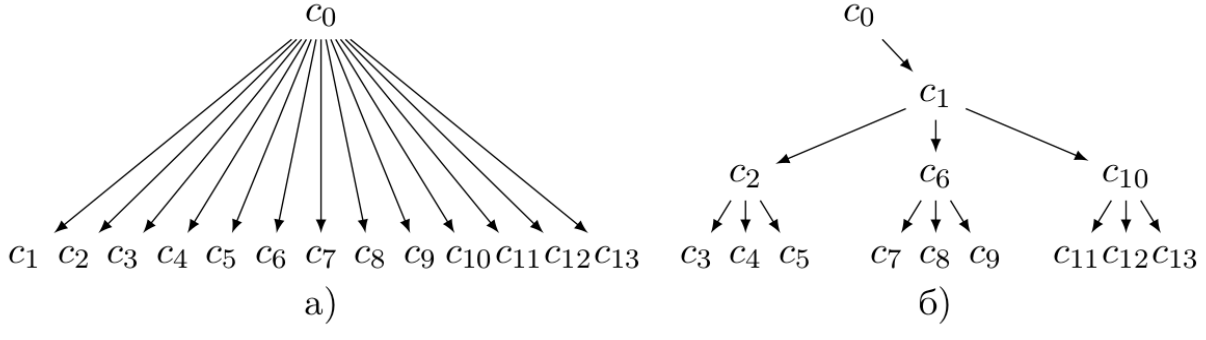


Рис. 1. Схема выполнения запросов к $n = 13$ вычислительным узлам: а – с помощью традиционных средств удалённого доступа; б – с помощью построения дерева запроса со степенью вершины $f = 3$

1.2. Сложность операции

В качестве примера рассмотрим дерево запроса со степенью вершины $f = 3$ представленное на рис. 1,б). Имеется запрос к $n = 13$ узлам. Обозначим список узлов в запросе как $c_{1,2,\ldots,13}$. Узел c_1 является корнем запроса. Он исключается из исходного списка узлов, а оставшуюся часть разделяет на $f = 3$ части $c_{2,\ldots,5}$, $c_{6,\ldots,9}$, $c_{10,\ldots,13}$. Если число узлов n в оставшейся части не кратно f, то оно округляется до ближайшего кратного в большую сторону. Это позволяет формировать дерево максимально правильной формы при любом числе узлов n. В каждой части выбирается головной узел. Для определённости, в качестве головного можно всегда выбирать первый узел данной части. Головные узлы c_2, c_6, c_{10}, в свою очередь разделяют списки поступивших узлов на $f = 3$ части и транслируют дальше. На нижнем уровне дерева списки вырождаются в отдельные узлы.

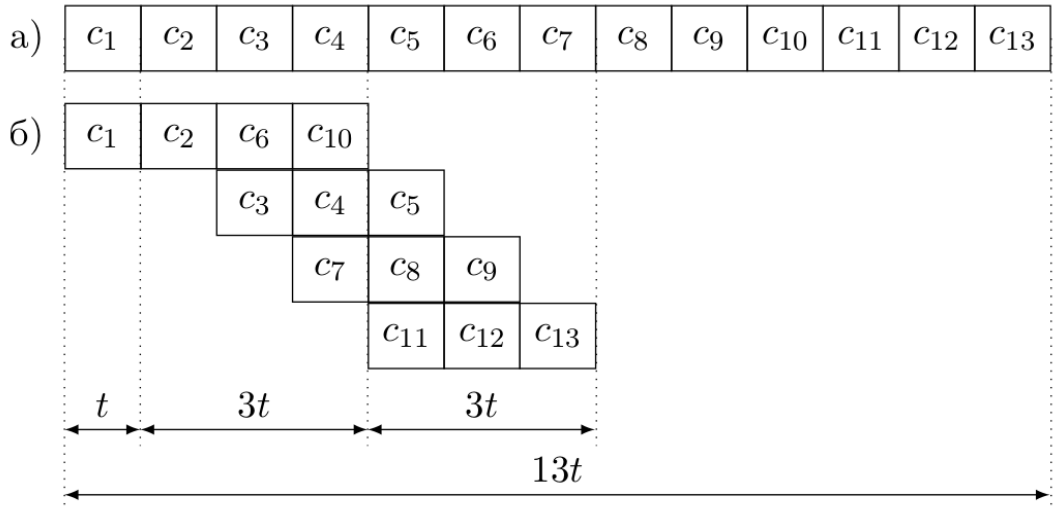


Рис. 2. Длительности запросов к $n = 13$ узлам $c_{1,\ldots,13}$: а – последовательный режим доступа; б – построение дерева запроса

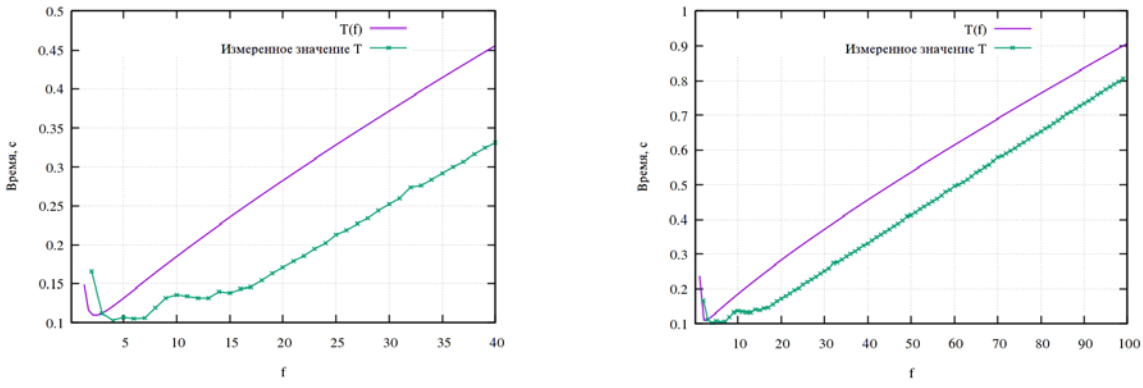
Длительность запроса t_i к каждому из узлов считаем одинаковой, т. е. $t_i = t$, $i = 1,\ldots,13$. Для последовательного выполнения запросов к 13 узлам потребуется время $13t$ (рис. 2, а). Если запросы от головного узла к дочерним узлам выполняются одновременно, длительность операции пропорциональна числу уровней в дереве, т. е. в данном случае $2t$. Однако каждый головной узел посылает запросы к своим дочерним узлам последовательно. Посылка запроса к узлам $c_{2,6,10}$ потребует $3t$ (рис. 2, б). Запросы к узлам $c_{11,12,13}$ начнутся не ранее завершения запроса к узлу c_{10} и займут $3t$. Ещё t потребуется для запроса к корневому узлу c_1. Поэтому, для посылки запросов к тем же узлам с помощью дерева, потребуется $3t + 3t + 1t = 7t$. Запросы к узлам $c_{3,4,5}$ и $c_{7,8,9}$ не влияют на длительность операции, т. к. они начнутся до отправки запроса первому узлу правого поддерева на нижнем уровне (узлы $c_{11,12,13}$ в данном случае) и продлятся не более $3t$.

Вообще, длительность операции определяется максимальной задержкой перед выполнением запроса к самому правому листу дерева запроса. Дерево имеет правильную форму по построению. Поэтому, задержка определяется степенью вершины f дерева и числом уровней h и составляет hft. Длительность всей операции складывается из длительности запроса к корневому узлу дерева и максимальной задержки, т. е. $T = (1+hf)\,t$. Максимальное число узлов в дереве высотой h и степенью вершины f есть $n = \left(f^{h+1} - 1\right) \Big/ \left(f - 1\right)$. Высота h дерева запроса $h = \log_f \left[n\left(f-1\right)+1\right] - 1$, откуда получаем

$$T = \left[1 + f \log_f \left[n\left(f-1\right)+1\right] - 1\right] t \,. \qquad (1)$$

1.3. Оптимальная степень вершины дерева запроса

Интерес представляет вопрос, какая степень вершины минимизирует длительность операции. Рассмотрим, например, поведение $T(f)$ при $n = 100$. Поведение кривой $T(f)$ и кривой, построенной из экспериментальных данных (рис. 3,а,б) для $n = 100$ имеет схожий характер. Видно, что сначала длительность операции снижается, достигает минимума и далее наблюдается рост. Снижение длительности T с ростом f можно объяснить тем, что уменьшается высота дерева. Несмотря на дальнейшее уменьшение высоты дерева, наблюдается дальнейший линейный рост T. Это можно объяснить тем, что растёт доля последовательных обменов между головным и дочерними узлами. При $f \geq n$ возникает вырожденный случай, когда высота дерева $h = 1$ и запросы выполняются последовательно.



а б

Рис. 3. $T(f)$ при $n = 100$ в диапазонах $2 \le f \le 40$ (а) и $2 \le f < 100$ (б)

Экспериментально полученное значение $f = 4$ (рис. 3) отличается от расчётного $f \approx 2$ (решение уравнения $T'(f) = 0$ даёт $f \approx 2{,}18$). Это, связано тем, что модель (1) не учитывает некоторые особенности. Реализация по протоколу UDP позволяет получать ответы от вычислительных узлов из начала заданного диапазона одновременно с окончанием отправки запросов узлам из хвоста диапазона. Длительность выполнения операции зависит от быстродействия оборудования, пропускной способности сети, и других факторов. Для более точного предсказания длительности выполнения, все эти факторы необходимо учесть в модели (1). Однако, даже в таком упрощённом виде, модель позволяет сузить область поиска оптимальной степени вершины.

1.4. Группировка ответных сообщений

Вычислительные узлы многопроцессорного вычислительного комплекса, как правило, имеют одинаковое оборудование и функционируют под управлением идентичного образа ОЗУ-резидентной операционной системы [6]. Подразумевается, что потенциальное число разных результатов выполнения удалённой команды (ответов) $r \ll n$. Поэтому хранить и передавать каждый результат отдельным блоком памяти избыточно. Одной из возможностей разработанного протокола является группировка одинаковых ответов на каждом уровне дерева запроса.

Каждый головной узел при приёме сообщений от своих дочерних узлов производит слияние одинаковых ответов и передаёт сжатый таким образом групповой ответ вверх по дереву. Каждому уникальному ответу ставится в соответствие список узлов. При слиянии очередного ответа от узла c_i, этот узел добавляется в нужный список. Слияние производится на каждом уровне дерева.

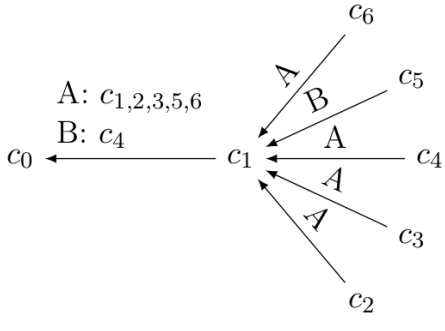


Рис. 4. Группировка одинаковых ответов при запросе к $n = 6$ вычислительным узлам $c_{1,\ldots,6}$

В результате массового запроса к большому числу узлов может получиться лишь небольшое число различающихся ответов. Например, при системном администрировании часто ожидаются

только два различных ответа: «хорошо» и «плохо». На рис. 4 показан запрос к $n = 6$ вычислительным узлам. Каждый узел возвращает ответ на вопрос: прошёл ли узел все внутренние проверки и готов ли он к приёму задачи на счёт. В этом случае возможными ответами являются только A (да, готов) и B (нет, не готов). Узлы из списка с ответом A являются «хорошими». Узлы из списка с ответом B подлежат дальнейшему анализу (в данном случае – узел c_5).

Таблица 1

Пример группировки ответов от узлов $c_{1,2,\ldots,6}$

До группировки	После группировки
A: c_1	A: $c_{1,2,3,4,6}$
A: c_2	B: c_5
A: c_3	
B: c_4	
A: c_5	
A: c_6	

Приведённый пример наглядно демонстрирует сокращение объёма данных без потери количества информации (табл. 1). Важным следствием группировки является то, что она позволяет представить результат при выдаче пользователю в более наглядном виде. Кроме того, группировка ответов сокращает объем передаваемых сообщений, увеличивает быстродействие протокола, сокращает длины копируемых блоков памяти.

1.5. Длительность ожидания ответного сообщения

После посылки запроса дочерним узлам, наступает цикл ожидания ответов. Ответы накапливаются и группируются по мере поступления. Необходимо определить момент, когда послать накопленные ответы на верхний уровень дерева запроса. Отправка результата группового запроса происходит либо по готовности, либо по тайм-ауту. Готовность наступает, если число опрошенных узлов равно числу ответивших. При наличии сбойных узлов готовность может не наступить. Поэтому дополнительным условием отправки является тайм-аут. Тайм-аут наступает, если длительность ожидания ответов превысила w циклов приёма сообщений системным процессом. (Под циклом приёма сообщений здесь понимается одна итерация ожидания данных на файловом дескрипторе входящих сообщений с помощью системного вызова poll()).

Минимальная длительность ожидания есть такое число циклов приёма, которое равно высоте дерева запроса $w = h$ (необходим как минимум один цикл приёма на каждый нижележащий уровень дерева). Высота поддерева зависит от уровня l данного узла в общем дереве запроса. Число циклов ожидания на уровне l составит $w(l) = \log_f \left[n(f-1)+1 \right] - 1$. Такое количество циклов ожидания обеспечивает с одной стороны достаточную длительность ожидания ответов, с другой – приемлемую отзывчивость при наличии сбойных узлов.

2. Реализация набора команд

2.1. Утилита командной строки

Утилита командной строки выполняет разбор аргументов командной строки, введённых пользователем, и формирует сообщение запроса. В процессе разбора определяется команда протокола, тип запроса (прямой или групповой), общие параметры (список узлов для передачи команды, степень вершины дерева запроса), индивидуальные параметры команды. Обычно утилита командной строки выполняется пользователем на инструментальном сервере. Запрос отправляется

корневому системному процессу на управляющем сервере. В качестве корневого может выступать любой системный процесс любого узла или сервера из состава МВК (рис. 5).

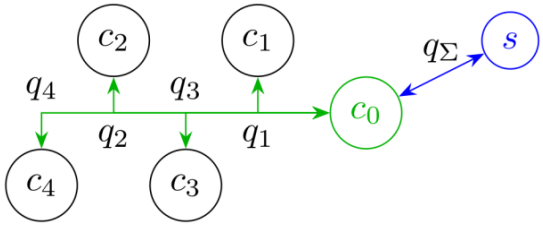


Рис. 5. Схема взаимодействия компонент: s – пользовательская утилита на сервере, c_0 – корневой системный процесс на вычислительном узле, $c_{1,2,3,4}$ – системные процессы на вычислительных узлах, q_i – прямые запросы и ответы, q_Σ – групповой запрос и ответ

При формировании запроса к множеству вычислительных узлов, утилита командной строки устанавливает флаг, свидетельствующий о том, что он является групповым. Корневой узел дерева запроса, получив сообщение с флагом «групповой запрос», присваивает ему идентификатор. Все передаваемые на нижележащие уровни, а также ответные сообщения помечаются присвоенным идентификатором.

2.2. Системный процесс

Системный процесс выполняет частичный разбор входящего сообщения. В процессе частичного разбора из сообщения извлекается информация: запрос это или ответ, команда, которую необходимо выполнить, прямое это сообщение или групповое, идентификатор запроса, запрашивающий узел для посылки ответа, список опрашиваемых узлов и т. д. Тело сообщения при этом не подвергается разбору и обработке.

Если частичный разбор сообщения показал, что это групповой запрос, из сообщения извлекается список, содержащий исходное множество узлов запроса. Текущий узел становится корнем группового запроса. Исходное множество узлов запроса делится на части. Для каждой части выбирается головной узел. Части формируют новые сообщения. Новые сообщения содержат части исходного множества узлов и тело исходного сообщения. Новые сообщения отправляются головным узлам. На головных узлах схема рекурсивно повторяется. Каждый из головных узлов становится новым корнем группового запроса. На каждом уровне число узлов запроса уменьшается.

Системный процесс одновременно обслуживает несколько групповых запросов. Каждый групповой запрос состоит из нескольких связанных друг с другом сообщений. Все сообщения в рамках запроса имеют одинаковый идентификатор. Групповой запрос подразумевает групповой ответ, который тоже состоит из нескольких связанных друг с другом сообщений. Системный процесс рассматривает каждое входящее сообщение независимо. Это может быть как часть группового запроса или ответа, так и прямое сообщение. Для отслеживания состояния группового запроса и накопления ответных сообщений предназначена таблица групповых запросов и ответов.

2.3. Таблица групповых запросов и ответов

Каждому вновь поступающему групповому запросу выделяется новый элемент таблицы групповых запросов и ответов. В выделенном элементе сохраняется информация о запросе: его идентификатор, список опрашиваемых узлов, тело транслируемого сообщения, степень вершины дерева запроса, количество и списки ответивших и не ответивших узлов и т. д. В табл. 2. в качестве примера приведены некоторые поля таблицы групповых запросов и ответов. Поле cmd определяет выполняемую команду протокола. Поле id определяет идентификатор запроса. Поле f (fanout) – степень вершины дерева запроса. Поля n_s и n_r – общее число узлов в запросе и число ответивших узлов. Поле head – список головных узлов нижележащего уровня дерева запроса.

При поступлении ответного сообщения на групповой запрос, системный процесс осуществляет поиск в таблице запросов и ответов. В качестве ключа поиска используется идентификатор запроса. Удачный поиск возвращает элемент таблицы с заданным идентификатором. Он обновляется в соответствии с вновь поступившей информацией из ответного сообщения. Ответные сообщения от всех опрошенных узлов в рамках заданного группового запроса накапливаются в этом элементе. По мере поступления ответов выполняется их группировка. Для этого, ответные сообщения организованы в бинарное дерево поиска. В качестве ключа поиска выступает тело ответного сообщения. Каждая вершина этого дерева содержит тело ответного сообщения и список узлов, приславших одинаковые ответы.

Таблица запросов и ответов организована в виде кольцевого буфера. Неудачный поиск в таблице говорит о том, что иной вновь поступивший групповой запрос в рамках кольцевого буфера занял место искомого элемента, и что ответное сообщение на групповой запрос сохранить негде. Такое сообщение отбрасывается. При большом числе запросов это штатная ситуация. Механизм кольцевого буфера предотвращает переполнение памяти при поступлении массовых групповых запросов. Используемый системным процессом объём памяти является константой. Компромисс между обработкой максимального числа запросов и количеством занятых ресурсов вычислительных узлов решается в пользу уменьшения потребления ресурсов за счёт отказа в обслуживании некоторых запросов. Число одновременно обрабатываемых групповых запросов определяется размером таблицы запросов и ответов. Это конфигурационный параметр системного процесса.

Таблица 2

Некоторые поля таблицы групповых запросов и ответов

cmd	*id*	*f*	n_s	n_r	head
CAT	816	5	40	40	*n* [101, 109, 117, 125, 133]
LS	817	3	86	86	n[102, 131, 160]
PS	814	4	49	49	n[103, 116, 128, 140]
GET	815	4	100	100	*n* [100, 115, 130, 144, 158, 172, 186]

При поступлении ответного сообщения на групповой запрос и удачном поиске элемента таблицы, выполняется проверка его готовности. Если число опрошенных узлов равно числу ответивших узлов, то формируется ответное групповое сообщение с данными этого элемента таблицы запросов и ответов. Сообщение отправляется на верхний уровень. Элемент таблицы помечается как свободный и может быть использован для нового группового запроса.

Системный процесс периодически сканирует таблицу групповых запросов и ответов и выявляет запросы с наступившим тайм-аутом (см. разд. 2.5). При наступлении тайм-аута, формируется групповой ответ из тех данных элемента таблицы, что есть в наличии, независимо от числа ответивших узлов. Ответ отправляется на верхний уровень дерева запроса. Элемент таблицы групповых запросов и ответов помечается флагом, сигнализирующим, что отправка всех накопленных в элементе данных была выполнена, он свободен и может быть использован вновь.

2.4. Формат сообщения

Для уменьшения времени передачи сообщений применяется бинарный формат сообщений. Перед передачей, необходимые структуры данных кодируются в выходное сообщение. После приёма выполняется декодирование. Структуры сообщений прямого запроса, группового запроса и ответа приведены на рис. 6. Каждое сообщение содержит обязательный заголовок. Заголовок сообщения содержит поля, позволяющие однозначно декодировать оставшуюся часть. Описанная в разделе 1.1 схема взаимодействия выступает в роли механизма маршрутизации сообщений. Данные (data[i]) при этом не меняются. Декодирование тела сообщения происходит только в конечном обработчике команды.

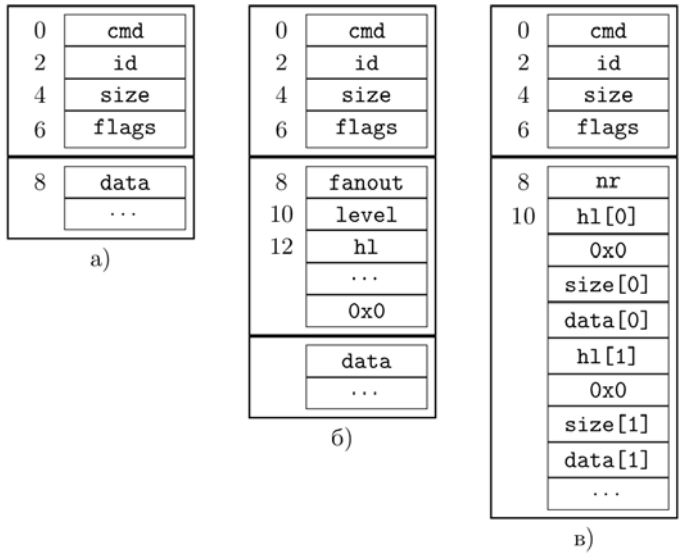


Рис. 6. Формат сообщений: а – прямой запрос/ответ, б – групповой запрос, в – групповой ответ

Структура прямого запроса и ответа совпадает (рис. 6,а). Она наиболее проста, и содержит только заголовок и данные.

Структура группового запроса (рис. 6,б) помимо заголовка и данных содержит информацию, необходимую для передачи запроса списку дочерних узлов. Поле fanout содержит степень вершины дерева запроса f. Поле level содержит уровень данного узла, начиная с корня дерева запроса. Поле увеличивается на единицу при формировании сообщения для передачи дочерним узлам. Поле hl содержит скобочно-префиксное представление списка узлов для передачи (см. 2.5). При обработке запроса данный список делится на части, которые кодируются в f сообщений для передачи дочерним узлам. Длина поля hl заранее неизвестна, она получается из общей длины сообщения за вычетом длины заголовка. Кроме того, для удобства декодирования список узлов оканчивается терминальным байтом 0x0.

Структура группового ответа (рис. 6, в) содержит nr групп. В каждую группу входит список узлов и тело сообщения. Тело сообщения data[i] длиной size[i] одинаково для всех узлов из списка hl[i]. Как и в случае группового запроса, для удобства декодирования каждый список узлов оканчивается терминальным байтом 0x0. Длины тел сообщений являются переменной величиной и кодируются полем size[i].

2.5. Списки узлов

Работа со списками узлов осуществляется с применением специальной схемы именования и префиксно-скобочное представление. Такое представление используется в частности в системе управления ресурсами Slurm [3]. Схема именования заключается в соединении префикса и логического номера для формирования имени узла (hostname). Например, вычислительные узлы $c_{1,2,\ldots,99}$ могут иметь имена n1, n2, …, n99. При перечислении более одного узла, префикс записывается один раз, и внутри скобок перечисляются отдельные узлы и диапазоны. Диапазоны образуются с помощью короткого тире. В качестве разделителя используется запятая «,». Перечисление узлов n1, n2, n3, n5 n7, n11, n12, n13 в префиксно-скобочном представлении выглядит так: n[1 – 3,5,7,11 – 13] и, соответственно, существенно короче. Как правило, чем больше узлов в перечислении, тем короче такое представление. Например, все узлы множества $c_{1,\ldots,99}$ представляются записью n [1 – 99].

В процедурах протокола манипуляция со списками узлов является частой операцией. В связи с этим разработана библиотека, содержащая функции объединения, проверки наличия определённого узла, пересечения, разделения на части и т. д.

3. Примеры использования

Параллельное выполнение удалённых команд по разработанному протоколу осуществляется посредством утилиты командной строки pep (Parallel Execution Protocol). Далее приведены примеры использования некоторых реализованных команд протокола. Команды реализованы с учётом накопленного опыта администрирования, поиска проблем выполняемых параллельных задач, сопровождения МВК.

Команда GET позволяет получить значение переменной системного процесса, информацию об операционной системе и т. п. Примером переменных являются self-pid — идентификатор системного процесса на вычислительном узле, verbose — уровень детализации отладочных сообщений системного процесса, port — номер порта для взаимодействия по сети, cpu-nr — число ядер, mem-total — объём ОЗУ, и т. д. Например, командой

> pep get mem-total cpu-nr nodes = n [100 – 199]

Можно получить число процессорных ядер и общий объём памяти на вычислительных узлах (переменные cpu-nr и mem-total):

```
cpu-nr  mem-total  hostlist
------  ---------  --------
2       2g         n107
2       4g         n[100 – 106,108 – 126,128 – 172,174 – 199]
2       4g         n127
4       4g         n173
```

Команда LS предназначена для проверки наличия файла в файловой системе и получения информации о нём. В условиях отладки и настройки многопроцессорной вычислительной системы, образ операционной системы для вычислительных узлов порой меняется несколько раз в день. Команда LS необходима для подтверждения того, что на всех вычислительных узлах присутствует тот или иной файл, что режимы доступа к файлу и его владельца одинаковы, что содержимое файлов идентично. Для проверки идентичности содержимого файлов на множестве узлов предназначен флаг v (version), который добавляет в вывод команды LS «версию» файла. В качестве версии файла используется sha1-сумма его содержимого. Например, командой

> pep ls path=/bin/busybox flags = v nodes = n [100 – 199]

Можно получить «версию» файла /bin/busybox:

```
version   hostlist
-------   --------
188b3ef4  n[100,102-199]
fda0a114  n101
```

Команда PS позволяет получить список процессов с множества вычислительных узлов. Команда имеет несколько параметров, позволяющих ограничить объем вывода, такие как comm = < substr > – командная строка выполняющегося процесса содержит подстроку < substr >, user = < uid > – выполняющийся процесс запущен от имени пользователя < uid >, flags = t (мнемоника time) – получить суммарную длительность выполнения процесса в режиме пользователя и в режиме ядра с момента запуска, и т. д. Например, команда

> pep ps comm = pepd flags = t nodes = n [100 – 104,200,300 – 309]

Выведет информацию об использованном процессорном времени процессом pepd (системный процесс на вычислительном узле):

```
t_u  t_s  t_r     ratio  comm  hostlist
---  ---  ------  -----  ----  --------
1s   1s   8d.23h  0.000  pepd  n [100,102 – 103,200,301,306 – 307]
1s   2s   8d.23h  0.000  pepd  n [101,104,300,302 – 305,308 – 309]
```

Здесь поля имеют следующие значения:

- t_u (мнемоника user) – общее процессорное время , потраченное в режиме пользователя;
- t_s (мнемоника system) – общее процессорное время, потраченное в режиме ядра операционной системы;

- t_r (мнемоника runtime) – астрономическая длительность выполнения процесса;
- ratio – отношение, показывающее, насколько интенсивно процесс потребляет процессорное время; для системных процессов отношение r должно быть как можно меньше; для прикладных кодов – как можно больше; типичным значением r для задачи, использующей все ресурсы вычислительного узла, является число процессорных ядер.

4. Обсуждение и сравнение с аналогами

Помимо упомянутых традиционных средств удалённого доступа на базе OpenSSH или RSH [7], таких как PDSH [2], для выполнения команд на вычислительных узлах можно воспользоваться утилитой srun из состава Slurm [4]. Утилита имеет широкие возможности по запуску MPI-задач, но может быть использована и для выполнения команд операционной системы. Благодаря использованию древовидного протокола, команда srun работает быстрее аналогичных средств. В разделах 4.1 и 4.2 приведены результаты сравнения разработанного средства удалённого доступа pep с утилитой pdsh, как наиболее распространённой, и утилитой srun, как наиболее быстрой.

Существует большое разнообразие других средств удалённого доступа. Так, например параллельный запуск команд операционной системы с помощью интерпретатора Python [8], или как ClusterSh [9]. Комплексные решения, например, такие как SaltStack [10] включают средства удалённого доступа как одну из функций. Существуют также утилиты, позволяющие группировать одинаковый вывод с множества вычислительных узлов на основе заданного шаблона, такие как dshbak [11]. Наконец, для разработки новых средств, можно воспользоваться низкоуровневыми библиотеками для организации взаимодействия по сети, работающие поверх BSD-сокетов, такие как ZeroMQ [12] или NanoMsg [13].

Утилиты пакета ClusterSh работают во многом подобно pdsh. Отличие состоит в использовании интерпретатора Python. Доступ осуществляется поверх SSH. Использование интерпретатора Python, как правило, ведёт к снижению быстродействия. Поверхностное тестирование данного средства показывает, что оно, как минимум не быстрее pdsh. Вообще, средства на базе SSH требуют порождения процессов. Реализация внутреннего обработчика невозможна даже для простого запроса. Любой запрос ведёт к выполнению одной или нескольких команд операционной системы. Это, ведёт к выполнению системных вызовов, выделению и освобождению памяти, и т. д. Как правило, такие средства реализуют последовательные схемы обменов, и плохо масштабируется на большое число узлов.

SaltStack предоставляет расширенные возможности управления множеством серверов различного назначения и конфигурации. Однако применимость его для управления вычислительным полем сравнительно низкая. Поскольку вычислительные узлы не содержат накопителей, загрузка происходит по сети. Одним из требований при формировании операционной системы для вычислительных узлов является минимальный размер загрузочного образа. Для работы протокола удалённого доступа SaltStack и средств из пакета ClusterSh необходим интерпретатор Python. Добавлять интерпретатор Python и комплект требуемых модулей на вычислительные узлы только для обеспечения работы средств удалённого доступа представляется нецелесообразным.

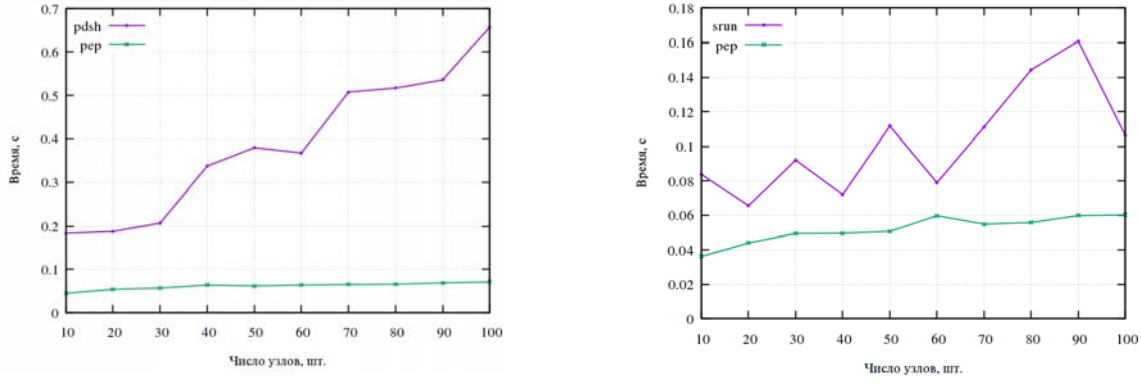
Низкоуровневые библиотеки, такие как ZeroMQ или NanoMsg упрощают программный интерфейс посылки и приёма сообщений, предлагают определённые шаблоны и схемы взаимодействия компонент. Например, обмены точка-точка, рассылка-подписка, и т. д. Шаблоны взаимодействия несколько ограничивают гибкость протокола, реализуемого с использованием таких библиотек. Кроме того, ограничивается выбор и нижележащих протоколов. Например, в ZeroMQ отсутствует возможность работать поверх UDP, который выгоден по двум причинам. Во-первых, он имеет минимальные накладные расходы. Во-вторых, примитивы send/recv проще всего реализовать в сетях, отличных от Ethernet. В связи с этим, необходимо дополнительное исследование применимости таких библиотек для реализации средств удалённого доступа в рамках МВК.

Приведённый перечень средств удалённого доступа составляет лишь небольшую часть общего их числа. Некоторые средства используются по традиции, другие являются стандартом de

facto. Часть средств являются подсистемами более обширных систем управления разнородной гетерогенной вычислительной сетью. Количество и разнообразие подобных средств можно объяснить тем, что большие вычислительные сети и многопроцессорные вычислительные комплексы часто являются штучными продуктами и имеют свои специфические требования.

4.1. Сравнение с утилитой pdsh

На (рис. 7,а) представлены результаты, измерения длительности выполнения команды PS протокола и эквивалентной конструкции с использованием сценария интерпретатора командной строки и команды параллельного выполнения pdsh, в зависимости от числа узлов. Число узлов варьировалось от 10 до 100. Каждый запуск повторялся 10 раз. Результат усреднялся. Измеренные длительности выполнения обозначены как «pdsh» и «pep». В обоих случаях выполняется поиск процесса с определённым именем на заданном множестве вычислительных узлов.



а б

Рис. 7. Длительность выполнения: а – команды протокола PS («pep») и эквивалентной команды на базе pdsh («pdsh») в зависимости от числа узлов, б – команды протокола EXEC («pep») и эквивалентной команды, запущенной с помощью утилиты srun системы управления заданиями Slurm («srun»), в зависимости от числа узлов; выполнялась команда операционной системы /bin/date

Видно, что в случае выполнения команды pdsh наблюдается линейный рост длительности выполнения с ростом числа узлов. В случае выполнения с помощью разработанного протокола, длительность выполнения растёт существенно медленнее.

4.2. Сравнение с утилитой srun из состава Slurm

На (рис. 7,б) представлены результаты измерения длительности выполнения команды протокола EXEC и эквивалентной команды, запущенной с помощью утилиты srun системы управления заданиями Slurm, в зависимости от числа узлов. Число узлов варьировалось от 10 до 100. Каждый запуск повторялся 10 раз. Результат усреднялся. Измеренные длительности выполнения обозначены как «srun» и «pep». Выполнялась команда операционной системы /bin/date. Для исключения влияния планировщика Slurm, используемые узлы были заранее выделены командой salloc.

Видно, что длительность выполнения команды /bin/date с помощью srun на любом числе узлов от 10 до 100 больше, чем длительность выполнения этой же командой с помощью разработанного протокола.

Несмотря на усреднение результата и выполнение команд на свободных от любых задач вычислительных узлах, длительность выполнения отдельных запусков в Slurm превышала среднюю в несколько раз. Можно объяснить это тем, что утилита srun «встроена» в большой программный комплекс Slurm. Запуск на выполнение задания сопровождается множеством действий, связанных с обеспечением дополнительных функций, а именно, перенаправление и сбор стандартного вывода

(stdout/stderr), подготовка окружения MPI (даже в случае – mpi = none), запись статистики и учётной информации в базу данных, и т. п. В связи с загруженностью вычислительных ресурсов и тем, что «плавание» длительности запуска заданий в Slurm не является целью данного исследования, более тщательного анализа и устранения причин отклонений длительности выполнения srun не проводилось. Более корректное сравнение возможно в случае «выделения» протокола передачи сообщений из исходного кода Slurm, исключив, таким образом, множества перечисленных факторов.

4.3. Недостатки и ограничения протокола

Реализованный протокол и средство удалённого доступа pep пока нельзя рассматривать полноценной заменой pdsh или srun, поскольку не решены следующие вопросы:

- отказоустойчивость при наличии сбойных узлов. Механизм автоматического обхода сбойных узлов реализован, но в настоящее время требует повторного выполнения запроса с данным идентификатором вручную.
- авторизация на вычислительных узлах. Этот вопрос требует тщательной проработки.

Разработанный протокол является скорее прототипом. Реализованные простые команды были добавлены в отладочных целях. В будущем на базе данного протокола планируется организовать более сложные команды сбора с вычислительных узлов профилировочной информации о выполняющихся процессах параллельной задачи, значений специальных регистров о текущем режиме потребления энергии, частоты процессора, реализация команды массового копирования файлов на вычислительные узлы, и т. д.

Заключение

Предложенная система, основанная на протоколе параллельного выполнения удалённых команд, направлена на повышение эффективности системного администрирования МВК. Отличительной особенностью протокола является формирование дерева запроса на заданном множестве вычислительных узлов, параллельная доставка и обработка сообщений, маршрутизация и группировка ответных сообщений независимо от их типа.

Формирование дерева запроса на множестве из n узлов снижает асимптотическую сложность операции запроса с O(n) до O(logn), где n — число вычислительных узлов. Параллельная доставка сообщений и выполнение команд уменьшает длительность отклика и ускоряет сбор ответов с вычислительных узлов. Группировка ответных сообщений позволяет сократить объём сетевого трафика, ускорить обработку, выдать результат в удобном для восприятия виде.

Реализованы обработчики часто используемых при системном администрировании команд операционной системы Unix/Linux (cat, grep, ls, ps и т. д.), что сократило число вызовов к ядру операционной системы и позволило избежать накладных расходов на порождение процесса операционной системы, выделение памяти и т. д. Протокол передачи отделён от обработчиков конкретных команд. Независимость схемы маршрутизации и группировки от типа сообщения упрощает реализацию обработчиков новых команд.

Разработанный протокол параллельного выполнения команд позволяет заменить многие часто используемые конструкции на базе сценариев интерпретатора командной строки. Выполнение команд при этом удобнее, быстрее, обладает меньшими накладными расходами.

Литература

1. Open SSH Project. [Electronic resource]. Mode of access: http://openssh.com. (accessed: 20.02.2015).

2. Pdsh – a multithreaded remote shell client. [Electronic resource]. Mode of access: http://sourceforge.net/projects/pdsh (accessed: 30.11.2016).

3. Авдеев М. П., Залялов Н. Н., Клиент-серверная система мониторинга ресурсов вычислительных модулей и серверов приложений // XIV Нижегородская сессия молодых ученых (технические науки). Нижний Новгород, 15 – 19 февраля 2009.

4. Slurm: Simple Linux Utility for Resource Management. [Electronic resource]. Mode of access: http://slurm.schedmd.com/slurm_design.pdf (accessed: 11.12.2016).

5. Postel J., User Datagram Protocol, RFC 768, USC/Information Sciences Institute, Marina del Rey, California, August 1980. [Electronic resource]. Mode of acces: http://tools.ietf.org/html/rfc768.

6. Ерёмин Е. В., Залялов Н. Н. ОЗУ-резидентная операционная система на базе ядра Linux, оптимизированная для высокопроизводительных вычислений // Вопросы атомной науки и техники. Серия: Математическое моделирование физических процессов. – 2015. – Вып. 2. – С. 69 – 77.

7. rsh – Clients and Servers for remote access commands. [Electronic resource]. Mode of acces: ftp://ftp.uk.linux.org/pub/linux/Networking/netkit (accessed: 12.09.2015).

8. Python – An interpreted, interactive, object-oriented programming language. [Electronic resource]. Mode of acces: http://www.python.org (accessed: 5.12.2015).

9. ClusterShell – Python Library and Tools. [Electronic resource]. Mode of acces: http://cea-hpc.github.com/clustershell (accessed: 20.07.2016).

10. SaltStack – Configuration Management Software and Remote Execution Engine: https://repo.saltstack.com (accessed: 10.11.2016).

11. dshbak – format output from pdsh command. [Electronic resource]. Mode of acces: http://pdsh.googlecode.com (accessed: 30.11.2016).

12. zmq – 0MQ lightweight messaging kernel. [Electronic resource]. Mode of acces: http://www.zeromq.org (accessed: 16.11.2016).

13. nanomsg – scalability protocols library. [Electronic resource]. Mode of acces: http://nanomsg.org (accessed 16.11.2016).

PROTOCOL FOR PARALLEL EXECUTION OF THE SYSTEM MANAGEMENT COMMANDS ON NODES OF A MULTIPROCESSOR SYSTEM

E. V. Eremin, N. N. Zalyalov

Russian Federal Nuclear Center –
All-Russian Research Institute of Experimental Physics, Sarov

The protocol and a set of commands for the remote access to computing nodes are presented. Conventional remote access tools, such as SSH, or PDSH, successively execute the request to n nodes and the asymptotic complexity of such process is $O(n)$. It is possible to reduce it to complexity $O(\log n)$ by using the so called request tree, which is the way to partition the original set of nodes into subsets to operate them concurrently. Execution results are grouped to reduce the amount of data and make it easier to analyze them. Internal command handlers (cat, grep, ls, ps, etc.) of OS UNIX/Linux have been implemented and allow eliminating overheads associated with creation of processes, memory allocation, etc.

Key words: request, response, message, utility, client, server, remote access, remote procedure call.