УДК 519.6

СИСТЕМА АДМИНИСТРИРОВАНИЯ ВЫЧИСЛИТЕЛЬНЫХ УЗЛОВ МНОГОПРОЦЕССОРНОГО КОМПЛЕКСА НА ОСНОВЕ ПРОТОКОЛА ПАРАЛЛЕЛЬНОГО ВЫПОЛНЕНИЯ УДАЛЕННЫХ КОМАНД

Е. В. Ерёмин, Н. Н. Залялов
(ФГУП "РФЯЦ-ВНИИЭФ", г. Саров Нижегородской области)

Описывается система администрирования вычислительных узлов многопроцессорного комплекса, основанная на оригинальном протоколе параллельного выполнения удаленных команд. В отличие от традиционных средств удаленного доступа, имеющих асимптотическую сложность $O(n)$, данный протокол имеет сложность $O(\log n)$.

Реализованы внутренние обработчики некоторых команд операционной системы UNIX/Linux, что позволяет устранить накладные расходы на порождение процессов, выделение памяти и т. д.

Ключевые слова: администрирование, протокол, параллельное выполнение, клиент, сервер, дерево запроса, утилита командной строки, удаленный доступ, удаленный вызов процедур.

Введение

Системное администрирование многопроцессорного вычислительного комплекса (МВК) основано на применении средств удаленного 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 счета.

Предварительная настройка вычислительного поля и системного программного обеспечения имеет сравнительно слабые ограничения по длительности. Диагностика и устранение проблем требуют от средств удаленного 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 удаленного доступа является запрос на удаленное 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 объем результата растут пропорционально числу вычислительных узлов. К тому же традиционные средства (SSH, PDSH) порождают цепочку процессов для выполнения каждой команды. Но, поскольку запросы выполняются на фоне производственного счета, средства удаленного доступа должны иметь минимум накладных расходов.

В данной работе предлагается протокол, в котором сокращение длительности запроса достигается выполнением команды в нескольких узлах одновременно. Сокращение объема данных происходит за счет группировки 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 уже существуют системы мониторинга, например [3].

Одним из способов выполнения команды на нескольких узлах одновременно является формирование дерева запроса. Дерево запроса как средство ускорения операций удаленного 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 компонентов 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 на 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-приемнику, в котором узлу-приемнику предписывается выполнить определенную команду. Прямой ответ есть ответное сообщение протокола, содержащее результат выполнения команды, которое отправляется узлом-приемником узлу-источнику. Групповой запрос и групповой ответ подразумевают выполнение определенной 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, ее параметры и т. д. В процессе интерпретации ответного сообщения из него извлекается результат выполнения команды.

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

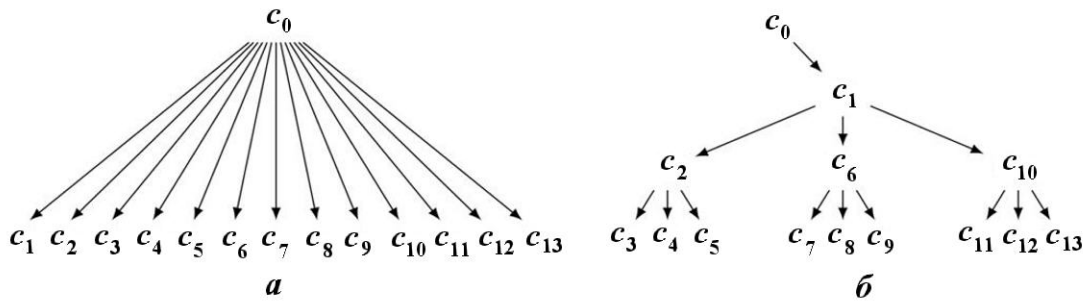

Рис. 1. Схема выполнения запросов к $n = 13$ вычислительным узлам: *а* — с помощью традиционных средств удаленного доступа; *б* — с помощью построения дерева запроса со степенью вершины $f = 3$

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, c_2,. . . ,c_{13}[1]. Узел c_0 не входит в этот список, на нем выполняется отправка запроса. Он исключается из исходного списка узлов, а оставшаяся часть разделяется на $f = 3$ части: c_2,. . . ,c_5; c_6,. . . ,c_9; c_{10},. . . ,c_{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 определенности в качестве головного можно всегда выбирать первый узел данной части. Головные узлы c_2, c_6, c_{10}, в свою очередь, разделяют указанным способом списки поступивших узлов на $f = 3$ части и транслируют их дальше. На нижнем уровне дерева списки вырождаются в отдельные узлы.

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 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, c_6, c_{10} потребует времени $3t$ (рис. 2, *б*). Запросы к узлам c_{11}, c_{12}, c_{13} начнутся не ранее завершения запроса к узлу c_{10} и займут также $3t$. Еще t потребуется для запроса к корневому узлу c_1. Поэтому для посылки запросов к тем же узлам с помощью дерева потребуется время $3t + 3t + 1t = 7t$. Запросы к узлам c_3, c_4, c_5 и c_7, c_8, c_9 не влияют на длительность операции, так как они начнутся до отправки запроса к первому узлу правого поддерева на нижнем уровне (узлу c_{11}) 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 $n = \left(f^{h+1} - 1\right) / (f - 1)$. Высота дерева запроса $h = \log_f\bigl(n\,(f-1)+1\bigr) - 1$. Отсюда получаем

$$T = \left[1 + f \log_f\bigl(n\,(f-1)+1\bigr) - 1\right] t. \tag{1}$$

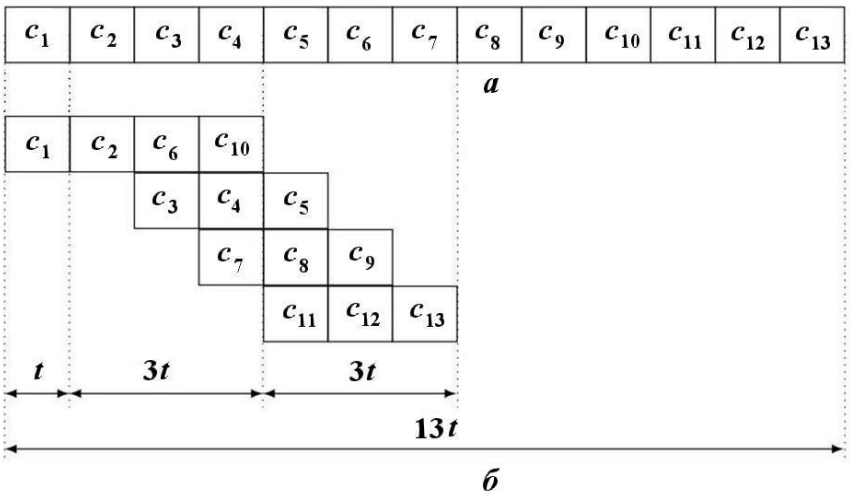

Рис. 2. Длительности запросов к $n = 13$ узлам c_1,. . . ,c_{13}: *а* — при последовательном режиме доступа; *б* — при построении дерева запроса

1.3. Оптимальная степень вершины дерева запроса. Интерес представляет вопрос, какая степень вершины дерева минимизирует длительность операции. Рассмотрим, например, поведение $T(f)$ при $n = 100$. Характеры поведения кривой $T(f)$ и кривой, построенной по экспериментальным данным (рис. 3, *а*, *б*), для $n = 100$ похожи. Видно, что сначала длительность операции снижается, достигает минимума, а далее наблюдается рост. Снижение длительности T с ростом f можно объ-

[1]Здесь и всюду далее узлы отождествляются с их логическими номерами.

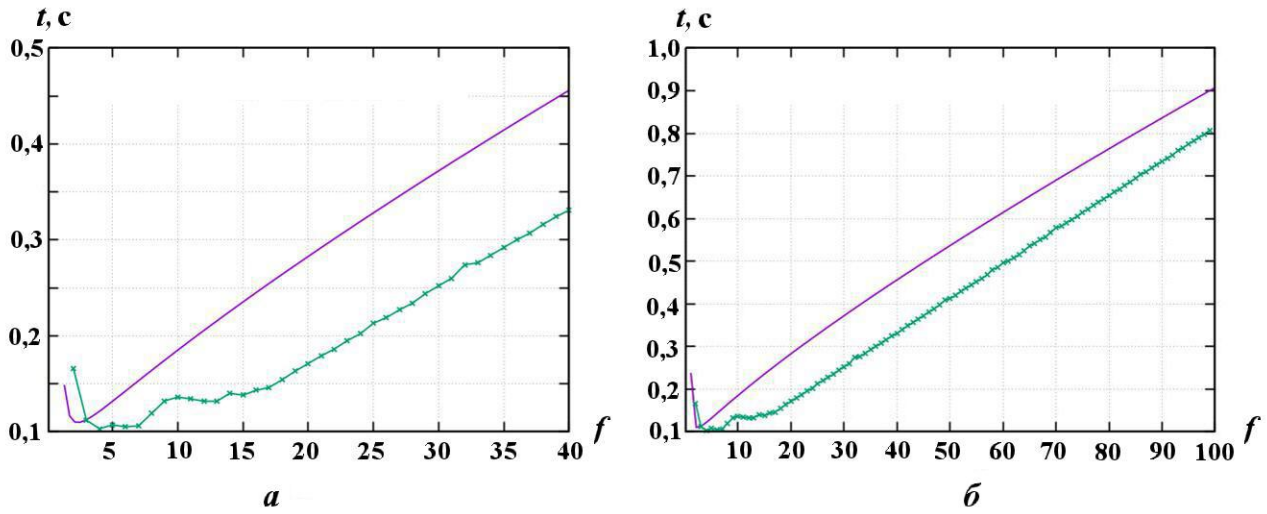

Рис. 3. $T(f)$ при $n = 100$ в диапазонах $2 \le f \le 40$ ($а$) и $2 \le f < 100$ ($б$): —— — расчет; –×– — эксперимент

яснить тем, что уменьшается высота дерева. Дальнейший линейный рост T с уменьшением высоты дерева можно объяснить тем, что растет доля последовательных обменов между головным и дочерними узлами. При $f \ge n$ возникает вырожденный случай, когда высота дерева $h = 1$ и запросы выполняются последовательно.

Экспериментально полученное значение оптимальной степени вершины дерева $f = 4$ (см. рис. 3) отличается от расчетного $f \approx 2$ (решение уравнения $T'(f) = 0$ дает $f = 2{,}18$). Это связано с тем, что модель (1) не учитывает некоторых факторов. Использование протокола UDP позволяет получать ответы от вычислительных узлов с начальными номерами из заданного диапазона одновременно с окончанием отправки запросов к узлам с номерами, которые находятся в конце 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 упрощенном виде модель позволяет сузить область поиска оптимальной степени вершины дерева.

1.4. Группировка ответных сообщений. Вычислительные узлы МВК, как правило, имеют одинаковое оборудование и функционируют под управлением идентичного образа ОЗУ-резидентной операционной системы [6]. Подразумевается, что потенциальное число разных результатов выполнения удаленной 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 приеме сообщений от своих дочерних узлов производит слияние одинаковых ответов и передает сжатый таким образом групповой ответ вверх по дереву. Каждому уникальному ответу ставится в соответствие список узлов. При слиянии с очередным ответом от узла c_i этот узел добавляется в нужный список. Слияние производится на каждом уровне дерева.

В результате группового 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 шести вычислительным узлам. Каждый узел возвращает ответ на вопрос, прошел ли он все внутренние проверки и готов ли к приему задачи на счет. 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).

Приведенный пример наглядно демонстрирует сокращение объема данных без потери количества информации (табл. 1). Важным следствием группировки ответов является то, что она позволяет выдавать результат пользователю в более наглядном виде. Кроме того, группировка ответов сокра-

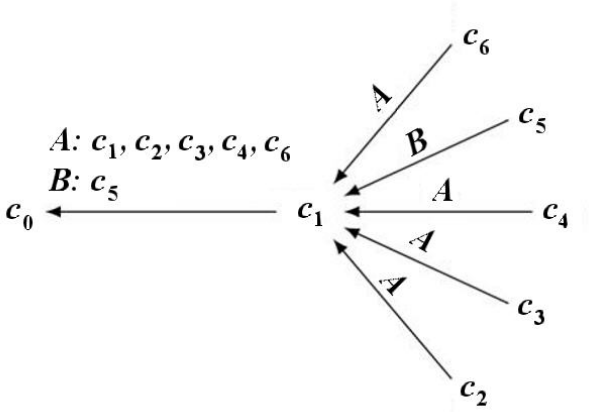

Рис. 4. Группировка одинаковых ответов при запросе к шести вычислительным узлам

щает объем передаваемых сообщений, увеличивает быстродействие протокола, сокращает длину копируемых блоков памяти.

Таблица 1

Данные из примера с шестью узлами без группировки и с группировкой ответов

Без группировки	С группировкой
A: c_1	A: c_1, c_2, c_3, c_4, c_5, c_6
A: c_2	B: c_5
A: c_3	
A: c_4	
B: c_5	
A: c_6	

1.5. Длительность ожидания ответного сообщения. После посылки запроса к 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. Посылка результата группового запроса происходит 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 приема сообщений системным процессом[2].

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

2. Реализация набора команд

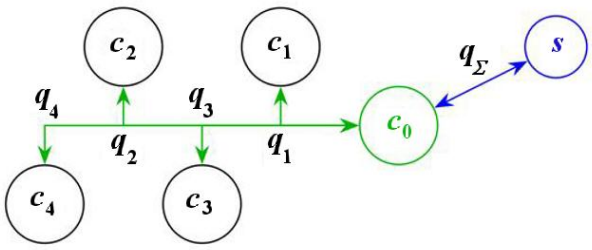

Рис. 5. Схема взаимодействия компонентов: s — пользовательская утилита на сервере; c_0 — корневой системный процесс на вычислительном узле; c_1,. . . ,c_4 — системные процессы на вычислительных узлах; q_1,. . . ,q_4 — прямые запросы и ответы, q_Σ — групповой запрос и ответ

2.1. Утилита командной строки. Утилита командной строки выполняет разбор аргументов командной строки, введенных 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 МВК (рис. 5).

[2]Под циклом приема сообщений здесь понимается одна итерация ожидания данных на файловом дескрипторе входящих сообщений с помощью системного вызова `poll()`.

При формировании запроса к множеству вычислительных узлов утилита командной строки устанавливает флаг, свидетельствующий о том, что запрос 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 ходе частичного разбора из сообщения извлекается следующая информация: является сообщение запросом или ответом, прямым или групповым; команда, которую необходимо выполнить; 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, в каждой из которых выбирается головной узел. Для каждой части узлов текущий узел формирует новые сообщения. Новые сообщения содержат части исходного множества узлов и тело исходного сообщения. Эти 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 может одновременно обслуживать 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 как часть группового запроса или ответа, так и прямое сообщение. Для отслеживания состояний групповых запросов 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 таблицы сохраняется информация о запросе: его идентификатор, список опрашиваемых узлов, список головных узлов, тело транслируемого сообщения, степень вершины дерева запроса, количества и списки ответивших и не ответивших узлов, структура данных с ответными сообщениями 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` — идентификатор запроса; `f` (fanout) — степень вершины дерева запроса; `n_s` и `n_r` — общее число узлов в запросе и число ответивших узлов; `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. При удачном поиске элемент таблицы с заданным идентификатором 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 виде бинарного дерева поиска. Здесь в качестве ключа поиска выступает тело ответного сообщения. Каждая

Таблица 2

Некоторые поля таблицы групповых запросов и ответов

cmd	id	f	ns	nr	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]

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

При поступлении ответного сообщения на групповой запрос и удачном поиске элемента таблицы выполняется проверка готовности ответа. Если число опрошенных узлов равно числу ответивших, то формируется ответное групповое сообщение с данными из 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. подразд. 1.5). 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 приема выполняется декодирование. Структуры сообщений прямых и групповых запросов и ответов приведены на рис. 6. Каждое сообщение содержит обязательный заголовок.

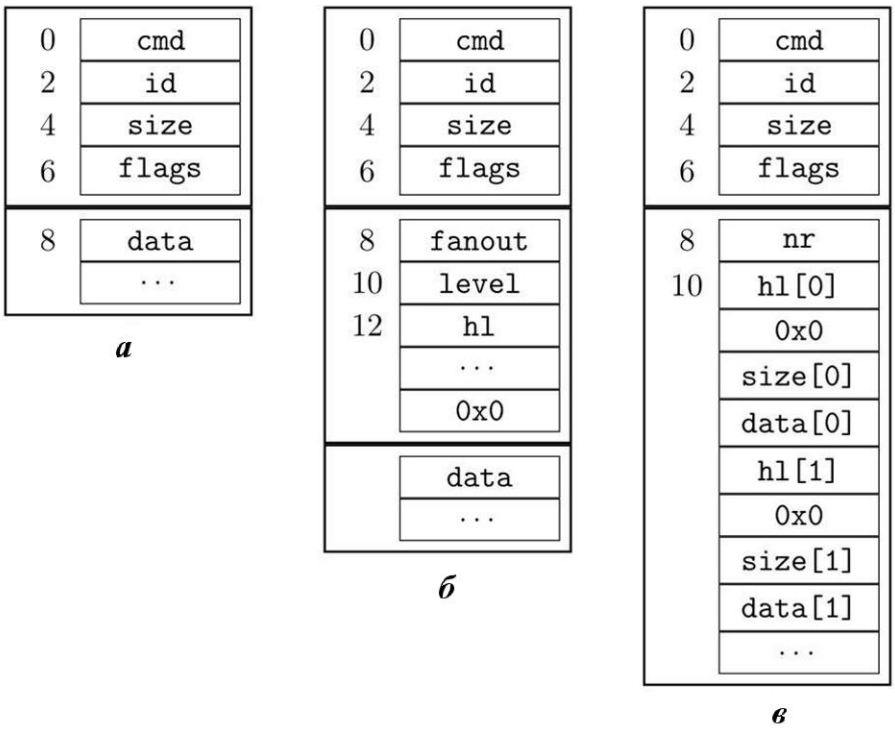

Рис. 6. Формат сообщений: *а* — прямые запрос и ответ; *б* — групповой запрос; *в* — групповой ответ

Заголовок содержит поля, позволяющие однозначно декодировать оставшуюся часть сообщения. Описанная в подразд. 1.1 схема взаимодействия программных компонентов выступает в роли механизма маршрутизации сообщений. Данные (`data`, `data[i]`) при этом не меняются. Декодирование тела сообщения происходит только в конечном обработчике команды.

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

Групповой запрос (см. 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. Значение `level` увеличивается на единицу при формировании сообщения для передачи дочерним узлам. Поле `hl` содержит скобочно-префиксное представление списка узлов для передачи (см. подразд. 2.5). При обработке запроса данный список делится на части, которые кодируются в f сообщений для передачи дочерним узлам. Длина поля `hl` заранее не известна, она получается из общей длины сообщения за вычетом длины заголовка. Для удобства декодирования список узлов заканчивается терминальным байтом `0` × `0`.

Групповой ответ (см. рис. 6, *в*) содержит `nr` групп. В каждую группу входит список узлов и тело сообщения. Как и в случае группового запроса, для удобства декодирования каждый список узлов заканчивается терминальным байтом `0` × `0`. Тело сообщения `data[i]` длиной `size[i]` одинаково для всех узлов из списка `hl[i]`. Длина тела сообщения является переменной величиной.

2.5. Списки узлов. Работа со списками узлов осуществляется с применением специальной схемы именования и префиксно-скобочного представления. Такое представление используется, в частности, в системе управления ресурсами Slurm [4]. Схема именования заключается в соединении префикса и логического номера для формирования имени узла (hostname). Например, вычислительные узлы c_1, c_2,...,c_{99} могут иметь имена `n1`, `n2`, ..., `n99`. При перечислении более одного узла префикс записывается один раз и внутри квадратных скобок перечисляются логические номера отдельных узлов и их 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 как `n[1-3,5,7,11-13]` и существенно короче. Как правило, чем больше узлов в перечислении, тем короче такое представление. Например, все узлы множества c_1,...,c_{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 определенного узла, пересечения, разделения на части и т. д.

3. Примеры использования

Параллельное выполнение удаленных команд по разработанному протоколу осуществляется посредством утилиты командной строки `pep` (Parallel Execution Protocol). Далее приведены примеры использования некоторых реализованных команд протокола. Команды реализованы с учетом 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. Переменными 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` — объем ОЗУ и т. д. Например, командой

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

можно получить число процессорных ядер (переменная `cpu-nr`) и общий объем памяти (`mem-total`) на вычислительных узлах с логическими номерами от 100 до 199:

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 нем. В условиях отладки и настройки МВК образ операционной системы для вычислительных узлов порой меняется несколько раз в день. Команда `ls` необходима для подтверждения того, что на всех вычислительных узлах присутствует файл с одним и тем же именем, режимом доступа, идентификатором владельца и содержимым. 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. Так, результатом команды

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

будет версия файла `/bin/busybox` на узлах с номерами от 100 до 199:

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

Команда `ps` позволяет получить список процессов на множестве вычислительных узлов. Команда имеет несколько параметров, позволяющих ограничить объем вывода: `comm=<substr>` — командная строка выполняющегося процесса содержит подстроку `<substr>`; `user=<uid>` — выполняющийся процесс запущен от имени пользователя `<uid>`; `flags=t` (time) — получить суммарную длительность выполнения процесса в режиме пользователя и в режиме ядра с момента запуска и т. д. Например, следующая команда предназначена для определения использованного процессорного времени процессом `pepd` (системный процесс) на вычислительных узлах из запрашиваемого списка:

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

Результат ее выполнения будет представлен в виде

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 `t_u` (user) — общее процессорное время, затраченное в режиме пользователя; `t_s` (system) — общее процессорное время, затраченное в режиме ядра операционной системы; `t_r` (runtime) — астрономическая длительность выполнения процесса; `ratio` — отношение, показывающее, насколько интенсивно процесс потребляет процессорное время; для системных процессов это отношение должно быть как можно меньше, для прикладных кодов — как можно больше. Типичным значением `ratio` для задачи, использующей все ресурсы вычислительного узла, является число процессорных ядер.

4. Обсуждение и сравнение с аналогами

Помимо упомянутых традиционных средств удаленного 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 подразд. 4.1

и 4.2 приведены результаты сравнения разработанного средства удаленного доступа `pep` с утилитой `pdsh` как наиболее распространенной и утилитой `srun` как наиболее быстрой.

Существует большое разнообразие других средств удаленного доступа, например параллельный запуск команд операционной системы с помощью интерпретатора Python [8] или ClusterSh [9], а также интегральные решения, включающие средства удаленного доступа как одну из функций, например SaltStack [10]. Существуют также утилиты, позволяющие группировать одинаковый вывод с множества вычислительных узлов на основе заданного шаблона, такие как `dshbak` [11]. Наконец, при разработке новых средств для организации взаимодействия по сети можно воспользоваться низкоуровневыми библиотеками, использующими на нижнем уровне BSD-сокеты, такими как ZeroMQ [12] или NanoMsg [13].

Утилиты пакета ClusterSh выполняются во многом подобно `pdsh`. Отличие состоит в применении интерпретатора Python и использовании на нижнем уровне `ssh`. Применение интерпретатора Python, как правило, ведет 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 ведет к выполнению одной или нескольких команд операционной системы. Это, в свою очередь, приводит к системным вызовам, выделению и освобождению памяти и т. д. Как правило, такие средства реализуют последовательные схемы обменов и плохо масштабируются 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 ее загрузочного образа.

Для работы протокола удаленного доступа SaltStack, как и для средств из пакета ClusterSh, необходим интерпретатор Python. Добавлять интерпретатор Python и комплект требуемых модулей на вычислительные узлы только для обеспечения работы средств удаленного доступа представляется нецелесообразным.

Низкоуровневые библиотеки, такие как ZeroMQ или NanoMsg, упрощают программный интерфейс посылки и приема сообщений, предлагают определенные шаблоны и схемы взаимодействия компонентов, например обмены *точка—точка*, *рассылка—подписка* и т. д. Шаблоны взаимодействия несколько ограничивают гибкость протокола, реализуемого с использованием таких библиотек. Кроме того, ограничивается выбор нижележащих протоколов. Например, в ZeroMQ отсутствует возможность использовать на нижнем уровне протокол UDP, в то время как UDP 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 низкоуровневых библиотек для реализации средств удаленного доступа.

Приведенный перечень средств удаленного доступа составляет лишь небольшую часть общего их числа; одни используются по традиции, другие являются стандартом de facto. Некоторые средства являются подсистемами более сложных систем управления разнородной гетерогенной вычислительной сетью. Количество и разнообразие подобных средств можно объяснить тем, что большие вычислительные сети и МВК часто являются штучными продуктами и имеют свои специфические требования.

4.1. Сравнение с утилитой `pdsh`. На рис. 7, *а* представлены результаты, измерения длительностей 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. В обоих случаях выполнялся поиск процесса с заданным именем на заданном множестве вычислительных узлов.

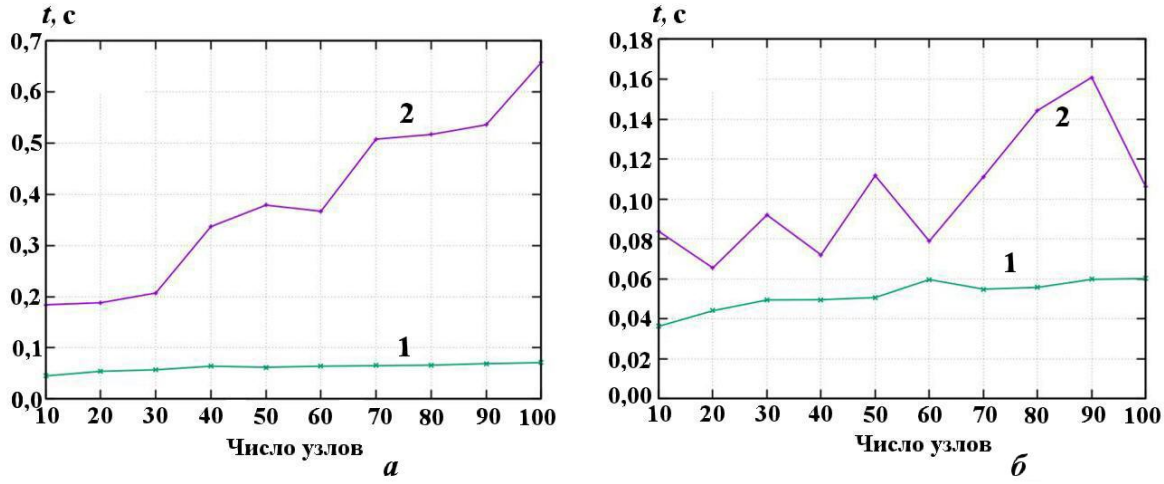

Рис. 7. Длительности выполнения команд в зависимости от числа узлов: *а* — команды протокола `ps` (1) и эквивалентной команды на базе `pdsh` (2); *б* — команды протокола `exec` (1) и эквивалентной команды, запущенной с помощью утилиты `srun` системы управления заданиями Slurm (2)

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

4.2. Сравнение с утилитой `srun` из состава Slurm. На рис. 7, *б* представлены результаты измерения длительностей выполнения команды протокола `exec` и эквивалентной команды, запущенной с помощью системы управления заданиями Slurm, в зависимости от числа узлов. Число узлов варьировалось от 10 до 100. Каждый запуск повторялся 10 раз. Результат усреднялся. Выполнялась команда операционной системы `/bin/date`. Для исключения влияния планировщика Slurm используемые узлы были заранее выделены командой `salloc`. Видно, что длительность выполнения этой команды с помощью `srun` на любом числе узлов от 10 до 100 больше, чем длительность ее выполнения 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: перенаправление и сбор стандартного вывода (`stdout/stderr`), подготовка окружения MPI (даже в случае `-mpi=none`), запись статистики и учетной информации в базу данных и т. п. В связи с тем, что непостоянство длительности запуска заданий в Slurm не являлось целью данного исследования, 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 с исключением таким образом множества перечисленных факторов.

4.3. Недостатки и ограничения протокола. Реализованный протокол и средство удаленного доступа `pep` пока нельзя рассматривать как полноценную замену `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, реализовать команды массового копирования файлов на вычислительные узлы и т. д.

Заключение

Предложенная система администрирования вычислительных узлов МВК, основанная на протоколе параллельного выполнения удаленных команд, направлена на повышение эффективности системного администрирования. 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 $O(\log n)$, где n — число вычислительных узлов. Параллельные доставка сообщений и выполнение команд уменьшают длительность отклика и ускоряют сбор ответов с вычислительных узлов. Группировка ответных сообщений позволяет сократить объем 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 ядра операционной системы и позволило избежать накладных расходов на порождение процесса операционной системы, выделение памяти и т. д. Протокол передачи отделен 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.

Список литературы

1. Open SSH Project. http://openssh.com.
2. PDSH — A Multithreaded Remote Shell Client. http://sourceforge.net/projects/pdsh.
3. *Авдеев М. П., Залялов Н. Н.* Клиент-серверная система мониторинга ресурсов вычислительных модулей и серверов приложений // 14-я Нижегородская сессия молодых ученых (технические науки). Нижний Новгород, 15—19 февраля 2009 г.
4. Slurm: Simple Linux Utility for Resource Management. http://slurm.schedmd.com/slurm_design.pdf.
5. *Postel J.* User Datagram Protocol RFC 768. USC/Information Sciences Institute. Marina del Rey, California. August 1980. 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. Сер. Математическое моделирование физических процессов. 2015. Вып. 2. С. 69—77.
7. RSH — Clients and Servers for Remote Access Commands. ftp://ftp.uk.linux.org/pub/linux/Networking/netkit.
8. Python — An Interpreted, Interactive, Object-Oriented Programming Language. http://www.python.org.
9. ClusterShell — Python Library and Tools. http://cea-hpc.github.com/clustershell.
10. SaltStack — Configuration Management Software and Remote Execution Engine. https://repo.saltstack.com.
11. `dshbak` — Format Output from `pdsh` command. http://pdsh.googlecode.com.

12. ZMQ — 0MQ Lightweight Messaging Kernel. http://www.zeromq.org.
13. NanoMsg — Scalability Protocols Library. http://nanomsg.org.

Статья поступила в редакцию 29.06.17.

MANAGEMENT SYSTEM FOR THE COMPUTATIONAL NODES OF THE MULTIPROCESSOR COMPLEX ON THE BASIS OF THE PROTOCOL OF THE CONCURRENT PROCESSING OF THE REMOTE COMMANDS / E. V. Eremin, N. N. Zalialov (FSUE "RFNC-VNIIEF", Sarov, Nizhny Novgorod region)

Management system for computational nodes of a multiprocessor complex based on the original protocol of the concurrent processing of the remote commands is described. Opposite to the traditional remote access means with asymptotic complexity $O(n)$ this protocol has $O(\log n)$ complexity.

Internal handlers of some commands of UNIX/Linux operating system are realized that allow eliminating overheads of process recreation, memory allotment etc.

Keywords: management, protocol, concurrent processing, client, server, inquest tree, command line utility, remote access, remote procedure call.