УДК 004.451

ОЗУ-РЕЗИДЕНТНАЯ ОПЕРАЦИОННАЯ СИСТЕМА НА БАЗЕ ЯДРА Linux, ОПТИМИЗИРОВАННАЯ ДЛЯ ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ ВЫЧИСЛЕНИЙ

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

Описываются основные моменты создания ОЗУ-резидентной операционной системы на базе ядра Linux, предназначенной для управления функционированием бездисковых вычислительных узлов мультипроцессорного вычислительного комплекса. Даются описания процедур формирования системы с возможностью изменения состава программного обеспечения путем добавления и удаления программных пакетов, формирования различных образов операционной системы для разных частей гетерогенного вычислительного комплекса. Детализированы процессы двухступенчатой нисходящей загрузки по коммуникационной сети вычислительного комплекса, ускоряющей загрузку одного вычислительного узла, и древовидной загрузки, ускоряющей загрузку всего комплекса. Приведены некоторые оценки влияния операционной системы на *параллельную* задачу, полученные с помощью локальных тестов, коллективных обменов и данных профилирования. Сделан краткий обзор аналогов.

Ключевые слова: мультипроцессорный вычислительный комплекс, ОЗУ-резидентная операционная система, ядро Linux, нисходящая загрузка, профилирование, параллельная задача, горячая точка, шум операционной системы.

Введение

Функционирование операционной системы (ОС) на вычислительных узлах мультипроцессорного вычислительного комплекса (МВК) должно удовлетворять некоторым специфическим требованиям. Одно из них — минимизация "посторонней" активности ОС. Примерами такой активности являются активизация системного процесса системы управления заданиями, синхронизация астрономического времени, администрирование журнальных файлов. Такого рода активность, в частности, влияет на длительность синхронизации большого числа процессов *параллельной* задачи. Другое требование налагает ограничения на потребление аппаратных и программных ресурсов: объема оперативной памяти, процессорного времени, пропускной способности сетевых интерфейсов и т. п. Кроме того, для бездисковых вычислительных узлов ОС и ее инфраструктура должны обладать процедурами нисходящей загрузки и инициализации посредством коммуникационной сети. И наконец, компоненты системы программирования вычислительного узла должны соответствовать системе программирования инструментальных серверов, обеспечивая тем самым совместимость среды разработки со средой исполнения пользовательских приложений.

ОЗУ-резидентная ОС (ОРОС) для вычислительных узлов МВК разработана с учетом указанных требований.

Состав и формирование

Назначение ОРОС — выполнение параллельных задач. Параллельная задача состоит из множества процессов, определенным образом распределенных по вычислительным узлам. Помимо процессов параллельной задачи, на каждом вычислительном узле также выполняются системные процессы: инициализация задачи и контроль ее статуса, проверка состояния вычислительного узла, периодическая синхронизация времени и пр. Чем меньше системных процессов выполняется на вычислительном узле, тем мень-

ше посторонняя активность ОС и тем выше доля процессорного времени, объема оперативной памяти и других ресурсов для параллельной задачи.

При попытке применить ОС общего назначения на вычислительных узлах возникают следующие проблемы:

- присутствует большое количество системных процессов, выполняющих различные вспомогательные функции, бóльшая часть которых не нужна. Эти процессы часто зависят друг от друга. Корректное их исключение из ОС затруднено;
- число вычислительных узлов в МВК сравнительно велико. Поддержание согласованности состояния ОС на них — сложная задача системного администрирования.

Поэтому был определен состав ОРОС для вычислительных узлов и реализована процедура ее формирования. ОРОС формируется на базе дистрибутива Linux (например Scientific Linux [1]). Она включает оптимизированное для целевой аппаратной платформы ядро Linux, необходимые драйверы, специализированное компактное окружение командной строки BusyBox [2] и систему инициализации.

В целях оптимизации загрузки в ОРОС выделены две части — начальный и основной образы ОС, каждый из которых представляет собой сжатый архив с корневой файловой системой. Два архива различаются по размеру, составу и функциональности. Назначение начальной ОС в том, чтобы загрузить по сети образ основной ОС. Образ основной ОС обеспечивает базовую функциональность.

Архивы формируются автоматически на основе конфигурационного файла. Конфигурационный файл позволяет настраивать различные характеристики ОРОС: выбор ядра, схему формирования адресов сетевых интерфейсов, адреса серверов точного времени и информации о пользователях, состав ОС. Состав ОС определяется списком пакетов. Пакет представляет собой описание исполняемых и конфигурационных файлов, разделяемых библиотек и вспомогательных файлов, обеспечивающих некую функциональность.

Для разных частей вычислительного поля в гетерогенном МВК могут быть созданы различные конфигурации ОС. Эти части могут отличаться назначением, составом оборудования или архитектурой процессора. Часть узлов может содержать более полное окружение, например для отладки (gdb [3], strace [4], valgrind [5] и т. д.), а другая часть — максимально облегченный вариант для выполнения отлаженных задач с предоставлением им бóльшего объема оперативной памяти.

В ОРОС используется ядро Linux. Ядро Linux для ОРОС предварительно конфигурируется и собирается на загрузочном сервере. С этого сервера осуществляется загрузка вычислительных узлов. Ядро собирается с оптимизацией под конкретную аппаратную конфигурацию (процессор и набор устройств). При этом отключаются возможности, не используемые на вычислительных узлах. Отключаются такие подсистемы, как SELinux, стек IPv6 и т. д. Отключаются драйверы для большого количества отсутствующего оборудования, например контроллер жесткого диска и драйверы файловых систем, поскольку ОРОС размещается в оперативной памяти вычислительного узла. В результате сокращается объем кода и число потоков ядра, что уменьшает влияние ОС на выполнение параллельных задач.

Ядро Linux выполняет перепланировку процессов по прерыванию системного таймера. За счет уменьшения частоты системного таймера может быть достигнуто снижение накладных расходов ОС на вызов планировщика и возможные переключения контекста процессов. Допустимые частоты системного таймера оригинального ядра Linux находятся в пределах 100—1 000 Гц. После модификации для ОРОС была установлена частота таймера 20 Гц.

Загрузка

Загрузка ОРОС на вычислительный узел происходит по сети в два этапа.

На первом этапе на узел передается образ небольшой начальной ОС (initramfs). Ее назначение в том, чтобы получить сетевую информацию, сформировать конфигурационный файл и загрузить основную ОС. Образ начальной ОС имеет минимальный размер, поскольку его передача идет по относительно медленному протоколу TFTP [6]. Применение этого протокола обусловлено прошитым на сетевой карте первичным загрузчиком.

На втором этапе начальная ОС выполняет загрузку основной ОС по протоколу FTP [7]. Протокол FTP справляется с большим трафиком и числом клиентов.

Наличие одного сервера загрузки, с одной стороны, обеспечивает полную идентичность копий ОС на всех вычислительных узлах, но, с другой, может стать узким местом в случае загрузки большого числа узлов. Для ускорения загрузки большого числа узлов применяется древовидная схема (рис. 1). Сначала загружаются вторичные загрузочные серверы из состава вычислительного поля. Затем с каждого вторичного загрузочного сервера загружается своя группа узлов. Таким образом, нагрузка на первичный загрузочный сервер снижается, а скорость загрузки всего поля увеличивается.

Рассмотрим более подробно процесс инициализации ОРОС.

При инициализации начальной ОС выполняются следующие действия:

1. Монтируются виртуальные файловые системы ядра.
2. Определяется состав оборудования конкретного вычислительного узла. Аппаратный состав вычислительных узлов может незначительно отличаться друг от друга. В процессе работы вычислительного узла процедуры автоматического определения и конфигурирования устройств, а также загрузки дополнительных модулей ядра исключены.
3. Определяется активный сетевой интерфейс. Он конфигурируется согласно настройкам, переданным DHCP-сервером. Сетевая загрузка может осуществляться по любому из сетевых интерфейсов вычислительного модуля. Сетевая информация сохраняется в конфигурационном файле узла для дальнейшего использования.
4. Загружается архив с файловой системой основной ОС и осуществляется переключение на основную ОС.

На этапе инициализации основной ОС выполняются следующие действия:

1. При необходимости активируются дополнительные сетевые интерфейсы по схеме, указанной в конфигурационном файле вычислительного узла.
2. С использованием этого конфигурационного файла генерируются другие необходимые конфигурационные файлы (limits.conf, yp.conf и т. д.).
3. Запускаются службы журналирования системных событий, журналирования критических ошибок оборудования (MCE) с возможностью передачи сообщений на сервер контроля аппаратных метрик (CKAM) [8].
4. Выполняются настройка и инициализация BMC (Baseboard Management Controller).
5. Запускаются необходимые службы: удаленного доступа (telnet, rsh, rlogin, ssh), мониторинга программного состояния вычислительного узла ResM [9], сбора статистики использования обменов и ввода-вывода прикладной параллельной задачи STK [10].
6. Если узел является вторичным загрузочным узлом, NFS-сервером и т. д., на нем загружаются соответствующие дополнительные службы.

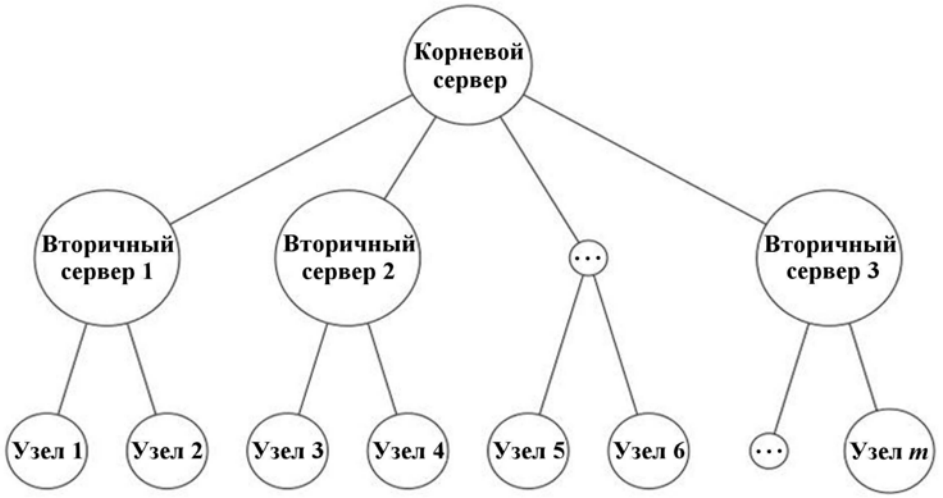

Рис. 1. Древовидная схема загрузки вычислительного поля

7. Осуществляются проверка доступности сетевых файловых систем и их монтирование.
8. Выполняется запуск узлового процесса системы управления заданиями JAM [11] или SLURM [12].

С этого момента вычислительный узел готов к выполнению процессов параллельной задачи.

Оценка влияния на параллельную задачу

Тестирование с помощью микротеста FTQ. Для исследования накладных расходов ОРОС использован микротест FTQ (Fixed Time Quantum) [13]. Идея теста заключается в подсчете количества итераций плотного цикла (busy loop) на одинаковых интервалах времени. Если активизируется какой-либо системный процесс, процессор переключается на него и число выполненных итераций оказывается меньше, чем на соседних интервалах.

При визуализации (рис. 2) по оси абсцисс откладывается номер интервала I, по оси ординат — число выполненных итераций *Count*. Из рис. 2, *а* видно, что при выполнении теста FTQ на ядре дистрибутива Scientific Linux количество итераций на разных интервалах имеет большой разброс. Иногда возникают существенные "провалы". Это означает, что процессор в режиме разделения времени используется несколькими процессами.

На рис. 2, *б* представлены результаты того же теста, но под управлением ОРОС с оптимизированным ядром. Диапазон изменения числа итераций цикла такой же, как на рис 2, *а*. Видно, что бóльшую часть времени на процессоре выполняется тест. Регулярные провалы вызваны обработкой прерываний от таймера. Таким образом, посторонние процессы не выполняются.

Профилирование операционной системы. Для выяснения причин вытеснения выполняемого приложения с процессора необходимо получить информацию о происходящих событиях: создании процесса (FORK), проецировании файла в память (MMAP) и т. д. В ядре Linux такую информацию предоставляет подсистема профилирования perf [14].

На базе подсистемы perf в ОРОС реализована подсистема профилирования параллельных задач rperf. Она предназначена для выявления причин системной активности, эффективности использования аппаратуры, наличия "горячих точек" в коде и т. д. Используя отметку времени, можно связать отклонения, демонстрируемые тестом FTQ, с идентификатором процесса, выполнявшегося в то же время.

На рис. 3 (см. также цветную вкладку) приведен результат выполнения теста FTQ при наличии постороннего процесса. По оси ординат слева отложен идентификатор порожденного процесса (*PID*), справа — число итераций цикла теста FTQ. Информация о создании процессов получена от системы профилирования с использованием события FORK. Посторонний процесс пе-

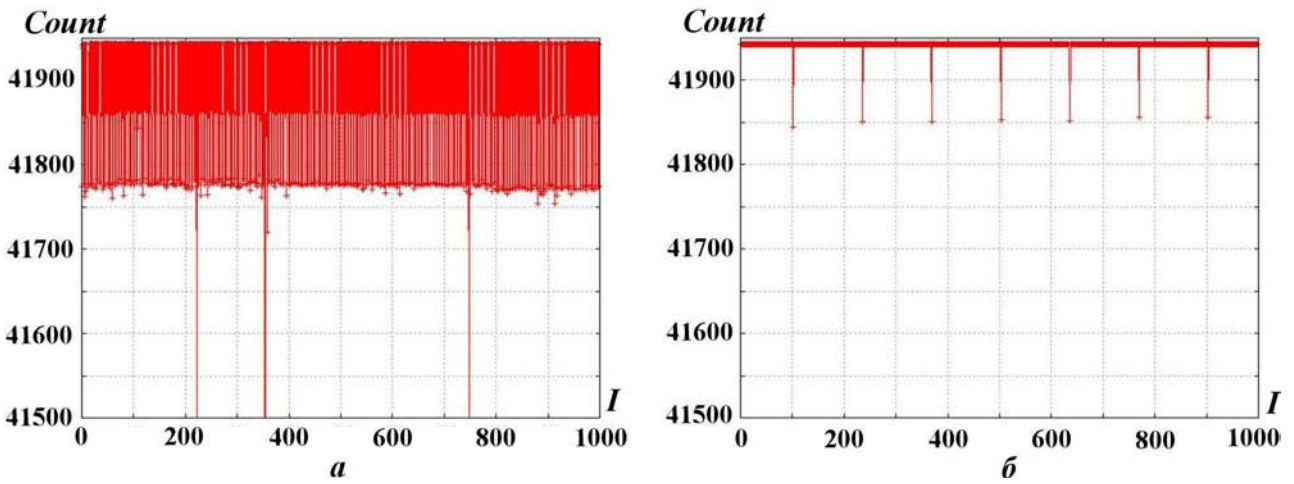

Рис. 2. Результат выполнения теста FTQ: *а* — на ядре из дистрибутива Scientific Linux; *б* — на оптимизированном ядре ОРОС

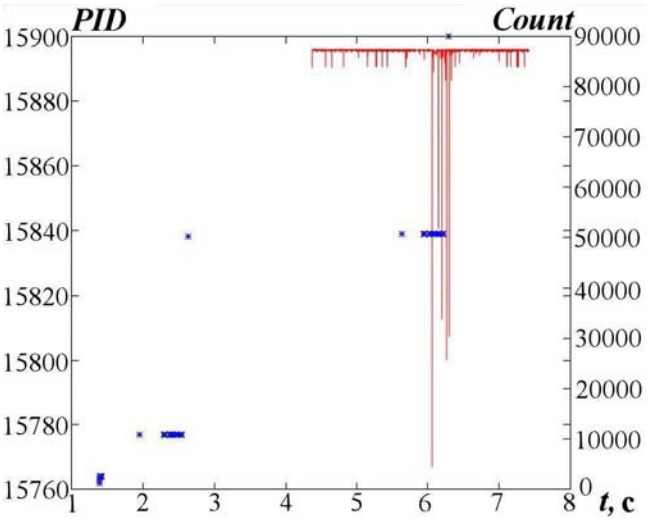

Рис. 3. Результат выполнения теста FTQ при наличии постороннего процесса: —— — число итераций; * — идентификатор процесса

риодически создавался, порождал потоки на все ядра вычислительного узла и выполнял некоторые вычисления (*шум*). Можно легко соотнести уменьшение числа итераций цикла теста FTQ с идентификатором постороннего процесса, мешающего выполнению теста.

Тестирование с помощью MPI-тестов. Для исследования того, как оптимизация ядра повлияла на выполнение параллельного приложения, был проведен ряд запусков тестовой программы, где измерялась длительность выполнения коллективной операции MPI. Коллективная операция является очевидной точкой синхронизации всех процессов в задаче. Сравнение велось с ядром из дистрибутива Scientific Linux 6. Посторонних процессов на узлы, занятые тестовой программой, не добавлялось.

В результате тестирования было получено отношение

$$r = \frac{T_d}{T_l},$$

которое показывает, во сколько раз длительность выполнения коллективных операций (Allgather, Allreduce, ...) под управлением дистрибутивного ядра Linux (T_d) больше, чем под управлением ядра ОРОС (T_l). Другими словами, если $r > 1$, то стандартная версия медленнее оптимизированной; если $r < 1$ — стандартная версия быстрее оптимизированной; если $r = 1$, различия версий по времени выполнения практически нет.

Отношение r для операции Allgather (рис. 4) почти во всех случаях больше единицы как для малых, так и для больших длин сообщений. Это отношение растет с ростом числа узлов в задаче (N). Таким образом, операция Allgather на оптимизированном ядре выполняется быстрее. Обмен сообщениями длиной 16 Кб на 512 узлах дает выигрыш около 2 раз.

Длительность выполнения более "легкой" операции Allreduce (рис. 5) не имеет ярко выраженной зависимости от описанной оптимизации ОС. На малых длинах сообщений заметен как проигрыш, так и выигрыш.

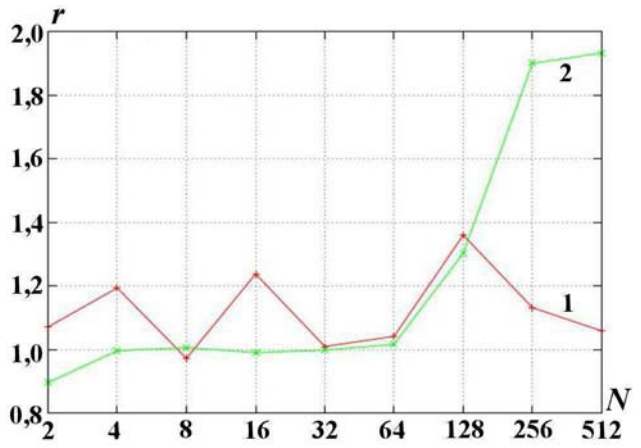

Рис. 4. Отношение r для операции Allgather: 1 — длина сообщений 1 Кб; 2 — длина сообщений 16 Кб

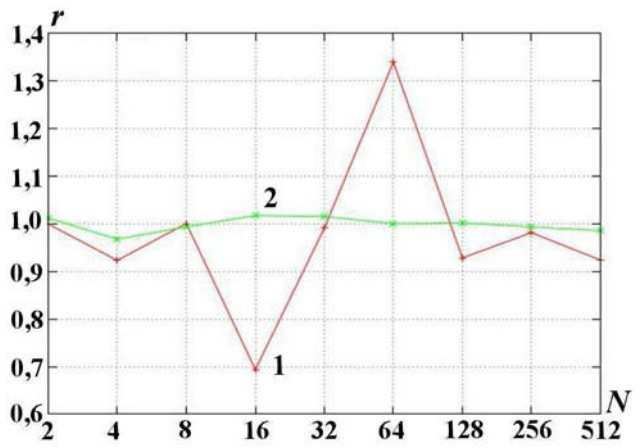

Рис. 5. Отношение r для операции Allreduce: 1 — длина сообщений 1 Кб; 2 — длина сообщений 16 Кб

Операции Allrtoall и Barrier демонстрируют поведение, подобное Allgather и Allreduce соответственно, поэтому оно здесь не приводится.

Обсуждение. Влияние посторонней активности на выполняющееся параллельное приложение показано во множестве работ (например [15—17]).

В [15] демонстрируется, что асинхронная активность в адресном пространстве пользователя вызывает увеличение длительности коллективных операций и она растет с числом узлов в задаче.

В [16] исследуется подобное влияние, но вызванное источником активности внутри ядра ОС. Показано, что даже очень незначительная активность в ядре может привести к существенному замедлению реальных приложений. Например, активность с частотой 1 000 Гц и длительностью всего лишь 25 мкс способна вызвать 30 %-ное падение производительности на 10 тыс. вычислительных узлов.

В [17] приводятся сведения о позитивном влиянии оптимизации ОС — исключения ненужных и настройки необходимых системных процессов (таймауты пробуждения) — на параллельные приложения. Также показано, что оптимизация алгоритмов выполнения коллективных операций может дать существенное ускорение (в 7 раз повышено быстродействие MPI_Allreduce).

Однако ускорение коллективных обменов не всегда приводит к существенному повышению эффективности параллельного приложения в целом. Так, вышеупомянутое семикратное ускорение MPI_Allreduce привело к увеличению производительности лишь на несколько (единицы) процентов.

Аналоги

Часто для больших вычислительных систем разрабатывается специальная ОС для вычислительных узлов. Такая ОС имеет ядро с минимумом функций, предназначенных только для управления аппаратурой. Большая часть функций по управлению ресурсами, процессами и файлами выполняется из режима пользователя специальным системным процессом или библиотекой. Применяется упрощенное планирование процессов с ограниченным числом возможных процессов на процессорном ядре. Зачастую нет поддержки виртуальной памяти, физические адреса проецируются в логические без изменений. Адресное пространство покрывается буфером TLB со страницами большого размера. Такая ОС имеет минимальное влияние на выполняемое приложение, минимальное потребление ресурсов и детерминированное поведение.

На МВК серии Blue Gene IBM применяется ОС CNK (Compute Node Kernel) [18]. Это компактное однопроцессорное и однопользовательское ядро, которое функцианирует на вычислительных узлах. В нем отсутствуют планирование процессов и поддержка файловых систем. Запросы ввода-вывода транслируются на выделенный узел ввода-вывода. Узлы ввода-вывода работают под управлением ОС на базе Linux, и на них монтируются файловые системы. С помощью специального процесса CIOD [19] запросы обрабатываются на узле ввода-вывода и результат отправляется обратно вычислительным узлам.

Для МВК Blue Gene в Аргонской лаборатории разрабатывается специализированная версия Linux — ZeptoOS [20]. Среди особенностей ZeptoOS можно выделить поддержку больших страниц, так называемую Big Memory [21], и специальный процесс ввода-вывода ZOID [22]. Используемый процессор не очень эффективно поддерживает механизм виртуальной памяти. С помощью больших страниц достигается увеличение низкой эффективности параллельных задач на данном типе процессора. Применение узлов ввода-вывода (ZOID) позволяет сократить число точек монтирования на файловых серверах. Узлы ввода-вывода выступают в качестве делителей нагрузки: у файловых серверов меньшее количество более активных клиентов.

Для МВК Cray в Сандийской лаборатории разрабатывается специализированная ОС Catamount [23]. Она аналогична CNK, но имеет немного большие возможности: поддерживаются многозадачность, многоядерность, управление памятью. Файловый ввод-вывод, как и в CNK, реализуется через узлы ввода-вывода.

Продолжением работ по Catamount в Сандийской лаборатории стал проект Kitten [24]. Работа направлена на устранение недостатков Catamount. Так, поддерживаются стандартные библиотека GLIBC и компилятор GCC, потоки POSIX Threads и OpenMP. Загрузка аналогична Linux, можно использовать стандартные средства для загрузки вычислительного поля по коммуникационной сети. Поддерживаются многоядерные процессоры, 64-битная архитекту-

ра x86. Одним из преимуществ разработчики считают открытость проекта [25].

Специализированные ядра являются аналогами ОРОС в том смысле, что решают ту же задачу — управления вычислительным узлом. В большинстве случаев эффективность выполнения параллельных задач под управлением специализированных ядер выше, чем под управлением Linux. Однако при использовании специализированных ОС возникают проблемы совместимости с оборудованием, сторонним системным и прикладным программным обеспечением, проблемы, связанные с удобством разработки параллельных задач, с сопровождением, выявлением и исправлением ошибок. Поэтому во многих МВК на вычислительных узлах применяются ОС на базе ядра Linux. На данный момент под управлением Linux функционируют 94 из 100 самых высокопроизводительных систем списка TOP500 [26]. Эффективность выполнения параллельных задач под управлением этих ОС может быть сравнимой с эффективностью под управлением специализированных ОС.

В качестве примера можно привести CNL (Compute Node Linux) [27], где выполнена работа по улучшению характеристик масштабируемости стандартного ядра Linux. CNL является частью проекта CLE (Cray Linux Environment). Авторы приводят результаты, свидетельствующие о том, что при интенсивной нагрузке на файловую систему Linux может быть предпочтительнее специализированного ядра [28].

Заключение

Очевидно, наличие посторонних системных процессов может увеличивать длительность выполнения параллельной задачи. Например, вытеснение предварительно считанного кода приложения из кэша инструкций приводит к необходимости дорогостоящих обращений к памяти при переключении контекста процессора. На одном вычислительном узле задержка может быть невелика, но она заметно замедлит работу параллельной задачи при синхронизации большого числа процессов. Такое замедление показано во множестве работ [15, 16].

Результаты тестов в данной работе показывают, что посторонняя активность оказывает измеримое влияние на выполняющийся процесс. Устранение всех возможных источников посторонней активности позитивно влияет на "тяжелые" коллективные обмены. Поэтому построение оптимизированной ОС для вычислительных узлов на базе ядра Linux и исследование ее влияния на параллельные приложения вполне оправданы.

ОРОС связывает системные компоненты, которые превращают набор отдельных вычислительных узлов в вычислительное поле. Компоненты эти разработаны как в РФЯЦ-ВНИИЭФ (СКАМ, ResM, STK, JAM), так и в других организациях (SLURM). ОРОС требует сравнительно небольших накладных расходов, оставляет параллельным задачам бóльшую часть оперативной памяти и процессорного времени, чем "полновесная" ОС, позволяет сократить длительность загрузки как отдельного узла, так и всего вычислительного поля, упрощает администрирование, позволяет использовать вычислительные узлы, не оснащенные локальными дисковыми накопителями.

Список литературы

1. Scientific Linux: An Enterprise Linux Rebuild Sponsored by Fermi National Accelerator Laboratory. https://www.scientificlinux.org/.
2. BusyBox: The Swiss Army Knife of Embedded Linux. http://www.busybox.net/.
3. GDB: The GNU Project Debugger. http://www.gnu.org/software/gdb/.
4. Strace: A System Call Tracer. http://sourceforge.net/projects/strace/.
5. Valgrind: An Instrumentation Framework for Building Dynamic Analysis Tools. http://valgrind.org/.
6. *Finlayson R.* RFC 906 — Bootstrap loading using TFTP // The Internet Engineering Task Force. June 1984. https://tools.ietf.org/html/rfc906.
7. *Postel J., Reynolds J.* File Transfer Protocol (FTP) // Ibid. October 1985. http://tools.ietf.org/html/rfc959.
8. *Богданов А. А., Залялов Н. Н., Ярулина А. А.* Система контроля аппаратных метрик. Архитектура, реализация // Вопросы атомной науки и техники. Сер. Математическое моделирование физических процессов. 2014. Вып. 4. С. 55—64.
9. *Авдеев М. П., Залялов Н. Н.* Клиент-серверная система мониторинга ресурсов вычисли-

тельных модулей и серверов приложений // 14-я Нижегородская сессия молодых ученых (технические науки). Нижний Новгород, 15—19 февраля 2009 г.

10. *Бартенев Ю. Г., Колпаков С. И., Киселёв А. Б. и др.* Программные средства STK для исследования эффективности выполнения параллельных приложений // Вопросы атомной науки и техники. Сер. Математическое моделирование физических процессов. 2011. Вып. 4. С. 72—81.
11. *Киселёв А. Б., Киселёв С. Н.* Система пакетной обработки заданий JAM // Там же. 2009. Вып. 4. С. 60—66.
12. SLURM: The Simple Linux Utility for Resource Management. http://slurm.schedmd.com.
13. *Sottile M., Minnich R.* Analysis of microbenchmarks for performance tuning of clusters // Proc. of the 6th IEEE Int. Conf. on Cluster Computing. San Diego, CA, September 2004. IEEE Press, 2004. P. 371—377.
14. Linux Profiling with Performance Counters. https://perf.wiki.kernel.org/index.php/Main_Page.
15. *Beckman P., Iskra K., Yoshii K. et al.* Benchmarking the effects of operating system interference on extreme-scale parallel machines // Cluster Computing. 2008. Т. 11, No 1. P. 3—16.
16. *Ferreira K. B., Bridges P., Brightwell R.* Characterizing application sensitivity to OS interference using kernel-level noise injection // Proc. of the 2008 ACM/IEEE Conf. on Supercomputing. IEEE Press, 2008. P. 19.
17. *Petrini F., Kerbyson D. K., Pakin S.* The case of the missing supercomputer performance: Achieving optimal performance on the 8,192 processors of ASCIQ // Proc. of the 2003 ACM/IEEE Conf. on Supercomputing. IEEE Press, 2003. P. 55—55.
18. *Moreira J. E., Brutman M., Castanos J. et al.* Designing a highly-scalable operating system: The Blue Gene/L story // Proc. of the 2006 ACM/IEEE Conf. on Supercomputing. ACM, 2006. P. 118.
19. *Moreira J. E., Almasi G., Archer C. et al.* Blue Gene/L programming and operating environment // IBM J. of Research and Development. 2005. Т. 49, No 2, 3. P. 367—376.
20. ZeptoOS: The Small Linux for Big Computers. http://www.mcs.anl.gov/research/projects/zeptoos/.
21. *Yoshii K., Iskra K., Naik H. et al.* Characterizing the performance of "Big Memory" on Blue Gene Linux // Int. Conf. "Parallel Processing Workshops ICPPW'09". IEEE, 2009. P. 65—72.
22. *Iskra K., Romein J. W., Yoshii K., Beckman P.* ZOID: I/O-forwarding infrastructure for petascale architectures // Proc. of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. ACM, 2008. P. 153—162.
23. *Kelly S. M., Brightwell R.* Software architecture of the light weight kernel, Catamount // Proc. of the 2005 Cray User Group Annual Tech. Conf. Sandia National Laboratories, 2005. P. 16—19.
24. Kitten: A Lightweight Kernel (LWK) Compute Node Operating System. https://software.sandia.gov/trac/kitten.
25. Kitten Lightweight Kernel. http://code.google.com/p/kitten/source/checkout.
26. TOP500: список самых высокопроизводительных МВК в мире. http://top500.org/statistics/sublist/.
27. *Wallace D.* Compute Node Linux: Overview, progress to date, and roadmap // Proc. of the 2007 Cray User Group Annual Tech. Conf. 2007. https://cug.org/5-publications/proceedings_attendee_lists/2007CD/S07_Proceedings/pages/Authors/Wallace-16C/Wallace_ paper.pdf.
28. *Wallace D.* Compute Node Linux: New frontiers in compute node operating systems // Proc. of the Cray User's Group. 2007. https://cug.org/5-publications/proceedings_attendee_lists/2007CD/S07_Proceedings/pages/Authors/Wallace-12B/Wallace-12B_paper.pdf.

Статья поступила в редакцию 12.12.14.

RAMdisk Linux-BASED OPERATING SYSTEM OPTIMIZED FOR HIGH-PERFORMANCE CALCULATIONS / E. V. Eremin, N. N. Zalyalov (FSUE "RFNC-VNIIEF", Sarov, Nizhny Novgorod Region).

The paper describes the basic milestones of the core resident Linux-based Operating System development to control the operation of diskless computer nodes of a multi-processor computer complex. We consider the procedures of the system building to enable the change of the software suit by adding and removing program packages, forming various operating system images for different parts of the heterogenic computer complex. We discuss in detail the processes of two-step downloading over the communication network of the computer complex, which accelerates the whole complex loading. We also provide several estimations of the operating system effect on the *parallel* code, which we have obtained using local tests, collective exchanges and profiling data. There is also a short review of related work.

Keywords: multi-processor computer complex, core resident Operating System, Linux core, descending loading, profiling, parallel code, hot spot, Operating Sistem noise.