КОНЦЕПЦИЯ ОРГАНИЗАЦИИ ХРАНЕНИЯ ДАННЫХ ДЛЯ ПАРАЛЛЕЛЬНОГО ВВОДА-ВЫВОДА В КЛАСТЕРНЫХ ВС.

Вдовикин О.И., Машечкин И.В.

Московский Государственный Университет им. М.В. Ломоносова, факультет ВМиК,
119992, Москва, Воробьёвы горы, +7-095-939-1988, +7-095-939-1789,
oleg@cs.msu.su,mash@cs.msu.su

This work describes an approach to the high-performance parallel I/O management in the multiprocessor computing environments. The author proposes a way for development a file system that specifically addresses the requirements of computing clusters. The work contains experimental results of practical approbation of given approach and a discussion for future research directions.

Введение

В большинстве задач, решаемых на кластерных ВС, основной обмен данными между параллельными ветвями осуществляется с помощью прикладных библиотек передачи сообщений, например, таких как MPI, а файловая система используются как средство для хранения входных и выходных данных.

Часть задач, например, задачи с визуальными средствами пост-обработки, разбивают данные по числу участвующих в счете ветвей. Каждая их них независимо использует либо индивидуальный файл, либо часть общего файла. Другим примером такого рода задач являются вычисления с сохранением контрольных точек. В этом случае каждая из участвующих в вычислении ветвей периодически сохраняет в файле свои текущие данные.

Для таких задач может оказаться возможным использование локально установленных в узлах вычислительного кластера устройств ввода-вывода. В таком случае их ветви могут обеспечивать практически полную локальность обменов с файлом внутри узла. Именно это предположение лежит в основе предлагаемой организации построения системы параллельного ввода-вывода в кластерных ВС.

Работа поддерживается грантом РФФИ № 02-01-00261.

Модель параллельного ввода-вывода

Параллельный ввод-вывод – высокоуровневый интерфейс, поддерживающий разбиение файловых данных между процессами, а также коллективные операции, поддерживающие сложные пересылки структур данных между памятью процессов и файлами [2].

Иллюстрацией возможной структуры «параллельного» файла может служить подход, использующийся в библиотеках MPI-IO [2]. В качестве хранимого в параллельном файле типа данных выступает один из элементарных типов MPI [1].

Каждая из ветвей параллельной программы определяет свой собственный тип данных – файловый тип данных – образованный либо из элементарного типа, либо из отнаследованного от него и возможно «пустот» (при этом их размер должен быть кратен используемому элементарному типу). Файловые типы данных задаются для каждой ветви отдельно, и в совокупности они формируют «запись» в параллельном файле. Рис. 1 иллюстрирует этот подход.

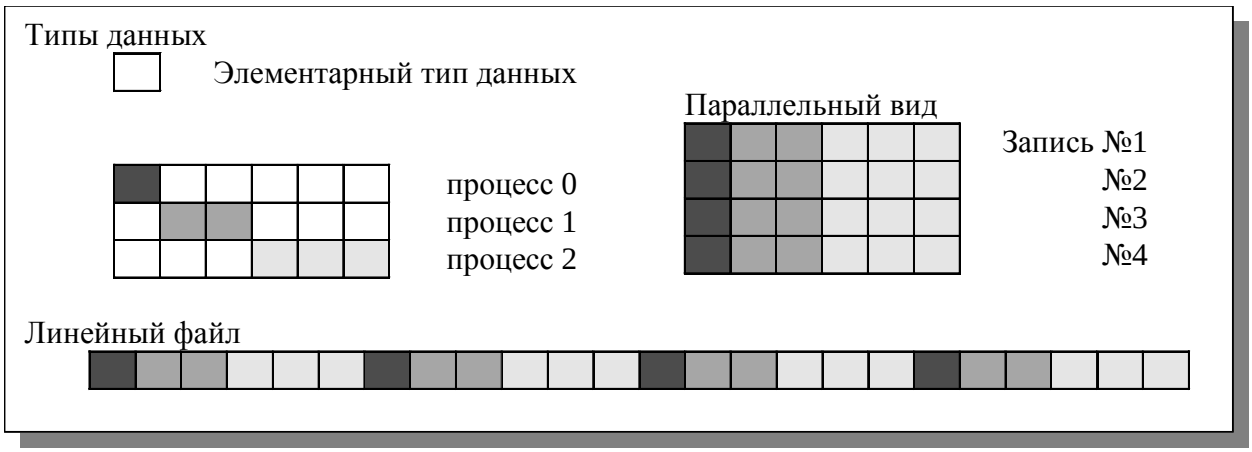

Рис. 1. Типы данных и представления файлов в MPI-IO

При такой организации «параллельный» файл в дальнейшем может быть представлен как последовательный, состоящий из отдельных записей. Все манипуляции над таким файлом производятся, как с линейным файлом и файловая система при его хранении более не учитывает его реальную структуру. Такой подход применяется в большом количестве параллельных файловых систем, например в PVFS [5], разработанной для кластерных вычислительных систем.

В параллельных системах такого рода различают как минимум два типа узлов – узлы ввода-вывода и вычислительные. Первые ответственны за формирование параллельной файловой системы, вторые – пользуются их сервисом. Как правило, эти два множества не пересекаются. Такой же подход к организации ввода-вывода обычно используется и при построении не кластерных мультипроцессорных систем. Рис. 2 представляет такую классическую схему.

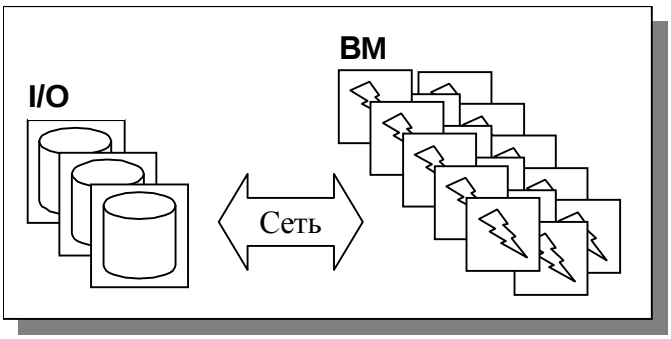

Рис. 2. Классическая схема организации ввод-вывод

При использовании такого подхода для хранения линейного файла обычно применяется прием, называемый stripping – разбиение файла на блоки фиксированного размера и круговое распределение блоков этого файла на узлы ввода-вывода.

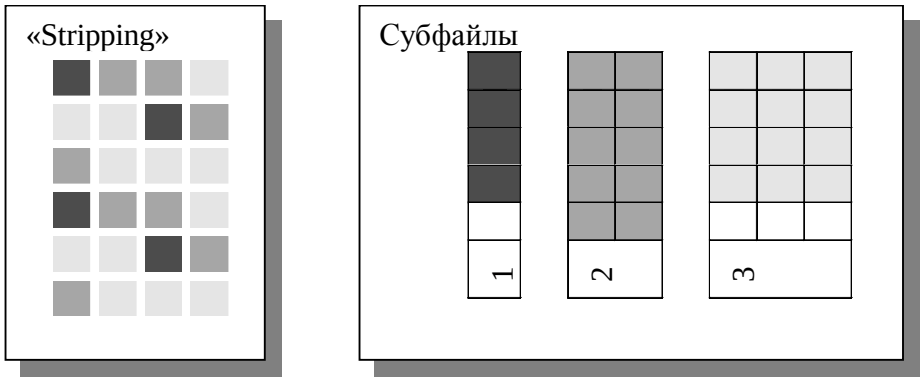

Рис. 3. Stripping и субфайлы

Для доступа к файлу в такой схеме может строиться запрос одновременно к нескольким узлам ввода-вывода. Это ведет к проблемам синхронизации при доступе к параллельному файлу, поддержании консистентности кэшей, файловых указателей и т.п. При таком подходе проблематично учесть разную производительность, участвующих в вычислении узлов: поскольку файл линеен, то запись в него происходит последовательно и т.о. быстрые узлы вынуждены дожидаться соседей.

Другой подход, призванный решить такую проблему – разбиение файла на субфайлы. При этом параллельный файл представляется как набор отдельных файлов (субфайлов), каждый из которых может обрабатываться независимо. Такой подход, например, используется в Vesta [3] и созданной на ее основе PIOFS [4]. Эти файловые системы используют классическую схему организации ввода-вывода, рассмотренную ранее, и stripping данных, но уже не на уровне файлов, а на уровне субфайлов. При использовании такого подхода упрощаются задачи по синхронизации ветвей параллельной программы. В большинстве случае она и вовсе становится не нужной [3].

Цель работы – построение системы, лишенной недостатка присущих большинству существующих параллельных файловых систем, при использовании их с подобными задачами – попытки хранения параллельных данных в последовательном виде, с дальнейшей их реконструкцией в параллельный.

Подход к организации параллельного ввода-вывода в кластерах

Общая структура системы. В предлагаемой параллельной системе комбинируются подходы, используемые в PVFS и Vesta: во-первых, для физического хранения файлов используются локальные диски узлов вычислительного кластера, во-вторых, используется разбиение файла на субфайлы. При этом не

выделяются отдельно узлы ввода-вывода и вычислительные – основная идея системы состоит именно в использования самих вычислительных узлов для выполнения операций с файлами.

Структура хранения данных тоже имеет существенные отличия от рассмотренных систем. В PVFS осуществляется низкоуровневое разбиение (stripping) последовательного файла на фрагменты, и дальнейшее его хранение на наборе различных узлов ввода-вывода. В Vesta, оперирующей субфайлами, происходит подобный процесс. При этом в обеих системах хранимые на узлах ввода-вывода данные сами по себе не несут никакой ценности – т.е. для получения целостных данных необходимо «собрать» информацию с группы узлов. В предлагаемой системе, весь субфайл (т.е. логически организованная структура), если позволяет разбиение данных параллельной задачи, хранится целиком только на одном узле ввода-вывода. Поскольку в данном случае субфайл – это, по сути, обычный файл, появляется возможность отказаться от разработки и использования специальной локальной файловой системы для хранения данных на узлах ввода-вывода. Более того, такая структура позволяет упростить не только хранение данных, но и массу связанных с этим задач, например их резервное копирование.

Другое немаловажное отличие – уровень, на котором обеспечивается параллелизм в файловой системе. Существующие системы параллельны «внутри», т.е. оперируют файловой системой как единым целым. Таким образом, метаданные «размазываются» по всему хранилищу файловой системы. Такой подход может быть использован только при условии 100% устойчивости системы к сбоям, как аппаратуры, так и программного обеспечения: разрушение части данных только на одном узле ввода/вывода может привести к краху всей файловой системы. В условиях использования кластерной системы, состоящей из сотен узлов, достичь такой стабильности практически невозможно. Поэтому, предлагается обеспечить параллелизм не на уровне средств файловой системы, а на уровне библиотек, например, таких как MPI-IO. При этом в качестве базовой файловой системы разработать и использовать распределенную файловую систему со средствами поддержки параллельных прикладных библиотек.

Организация данных. Логически в кластерной ВС файловое пространство предлагаемой ФС разбивается на три части:

1) дисковое пространство, доступное вычислительной задаче для хранения и обработки данных локально;

2) пространство, доступное вычислительной задаче для хранения и обработки данных и расположенное на других узлах вычислительного кластера;

3) метаданные, используемые для организации «параллельности» в файловой системе и используемые прикладной библиотекой.

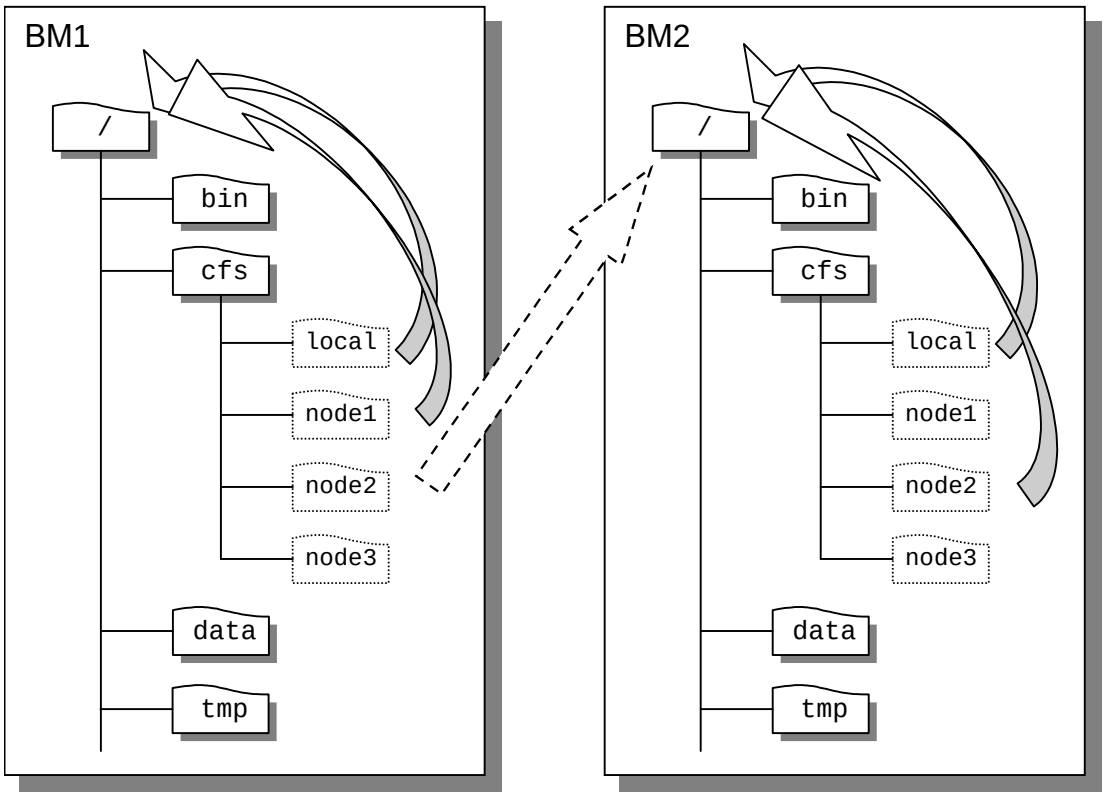

Рис. 4. Организация данных в системе

Для обеспечения прозрачности именования в рамках вычислительной системы используется выделенная и заранее известная точка «монтирования» файловой системы. В данном случае это каталог /cfs. Рис. 4 поясняет структуру первых двух частей системы.

Точка монтирования /cfs – корневая вершина распределенной файловой системы. Все ее непосредственные потомки «синтезируются» программным обеспечением. Для их именования используются имена узлов, входящих в вычислительную систему. На рисунке это node1, node2 и node3. Кроме этого, существует специальная вершина local, обозначающая часть распределенной системы, данные которой хранятся локально.

Программное обеспечение может быть построено таким образом, что с точки зрения операционной системы каталог local и каталог с именем, совпадающим с именем хоста, являются, по сути, символическими ссылками на корневую вершину / локального дерева каталогов. Остальные непосредственные потомки /cfs представляются каталогами. Т.о. после разрешения соответствия имени индексному дескриптору файла, все дальнейшие обмены с файлом, расположенным локально, идут с использованием «обычных» операций над локальными файловыми системами. Для работы с файлами, расположенными не на локальном узле, используется программное обеспечение распределенной файловой системы.

Следующим уровнем, который позволяет организовать «параллелизм» в файловой системе, является хранилище метаданных. В качестве такого хранилища в прототипе предлагается использовать сетевую файловую систему NFS (Network File System – сетевая файловая система, разработанная компанией Sun Microsystems), как наиболее стабильную и простую в использовании. Рис. 5 поясняет возможную структуру метаданных в файловой системе.

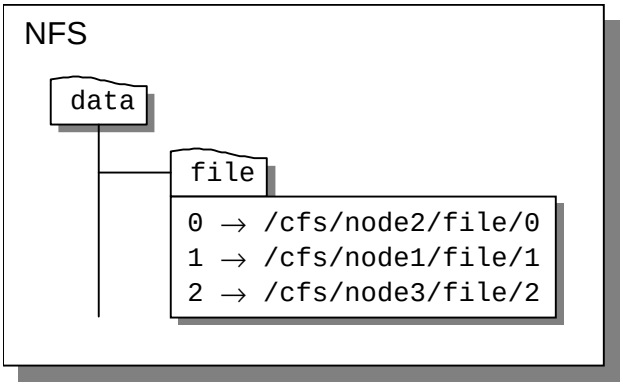

Рис. 5. Метаданные в параллельной файловой системе

Структура параллельного файла в метаданных представляется как набор субфайлов. При этом параллельному файлу в общей структуре хранения метаданных соответствует каталог, а субфайлам – содержимое этого каталога. Поскольку данные субфайлов хранятся не на NFS-сервере, а на вычислительных узлах кластера, вместо реальных файлов здесь хранятся только символические ссылки, указывающие реальное местоположение файла в распределенной файловой системе.

По сути, метаданные позволяют осуществить разбиение файла на субфайлы и указать прикладным библиотекам и программам места хранения этих субфайлов.

Программные компоненты системы. Программное обеспечение, формирующее предлагаемую параллельную файловую систему, разбивается на четыре части:

1) «виртуальная» файловая система, функционирующая как часть операционной системы в привилегированном режиме, обеспечивающая формирование непосредственных потомков вершины /cfs и осуществляющая диспетчеризацию обращений к ним;

2) сервер доступа к локальной файловой системе, обрабатывающий запросы удаленных клиентов;

3) клиенты удаленных файловых систем;

4) прикладные библиотеки параллельного ввода/вывода, формирующие «образ» файловой системы, как параллельной.

Поскольку многопроцессорные вычислительные кластеры, как правило, используются несколькими задачами одновременно, следует обратить особое внимание на создание наиболее «легковесной» серверной части: в случае, если задача обращается за данными, расположенными на вычислительном узле, в данный момент используемом другой параллельной задачей, влияние сервера доступа на производительность узла должна быть минимальной. Именно поэтому компонента сервера доступа должна работать в режиме ядра операционной системы, с тем, чтобы уменьшить количество переключений контекстов, использовать лишь буферный кэш оперативной памяти, осуществлять только асинхронную обработку дынных и т.п. Наибольшую нагрузку в этом случае должен нести клиент доступа, именно он должен, например, заниматься разбором имен.

С этой точки зрения, целесообразно в начальной реализации использовать наработки, имеющиеся в операционной системе для поддержки работы NFS-сервера. Семантика NFS-операций наилучшим образом соответствует выдвигаемым требованиям. Особую роль в этом играет изначальная концепция, заложенная в

протокол NFS – это сервер без состояния (state-less) и использование вызова удаленных процедур (RPC) для осуществления обменов.

Особенностью NFS-модели является и то, что при использовании данного протокола, в качестве транспорта сообщений может выступать сервис с негарантированной доставкой. Это позволяет, с одной стороны, не занимать сетевые каналы пакетами подтверждений, а с другой – максимально разгрузить NFS-сервер. Нормальное функционирование данной модели обусловлено тем, что клиент может повторять все операции, на которые он не получил подтверждения об исполнении, любое количество раз, при этом даже повторное их исполнение не приводит к каким-либо проблемам.

Прикладная библиотека параллельного ввода/вывода. Предполагается, что в качестве основной библиотеки параллельного ввода-вывода в параллельных программах будет использоваться библиотека MPI-IO. При этом основной упор будет нацелен не на получение собственного набора параллельных программных интерфейсов, которые потом будут использоваться для реализации набора функций MPI-IO, а на реализацию только набора интерфейсов ADIO [7], используемых в популярной реализации MPI MPICH [6], доступной для использования на большинстве кластерных вычислительных систем.

Одна из задач предлагаемой реализации – формирование и поддержание метаданных системы, поскольку именно они определяют «параллелизм» в файловой системе.

Обеспечение максимальной локальности обменов и поддержание метаданных осуществляется в реализации вызова ADIO_Open. Уже при открытии нового параллельного файла на запись, может происходить формирование всех необходимых субфайлов для параллельных ветвей программы по следующему сценарию:

1) переданная в качестве имени параллельного файла строка будет использоваться для создания одноименного каталога в области метаданных;

2) ветви параллельной программы, запущенные на нескольких узлах кластера, создают одноименные каталоги в корневой вершине распределенной файловой системы /cfs/local;

3) в этих каталогах, в области данных, создаются регулярные файлы с именами, соответствующими номеру ветви параллельной программы;

4) в каталоге, используемом для хранения метаданных, создаются символические ссылки на файлы в распределенной файловой системе.

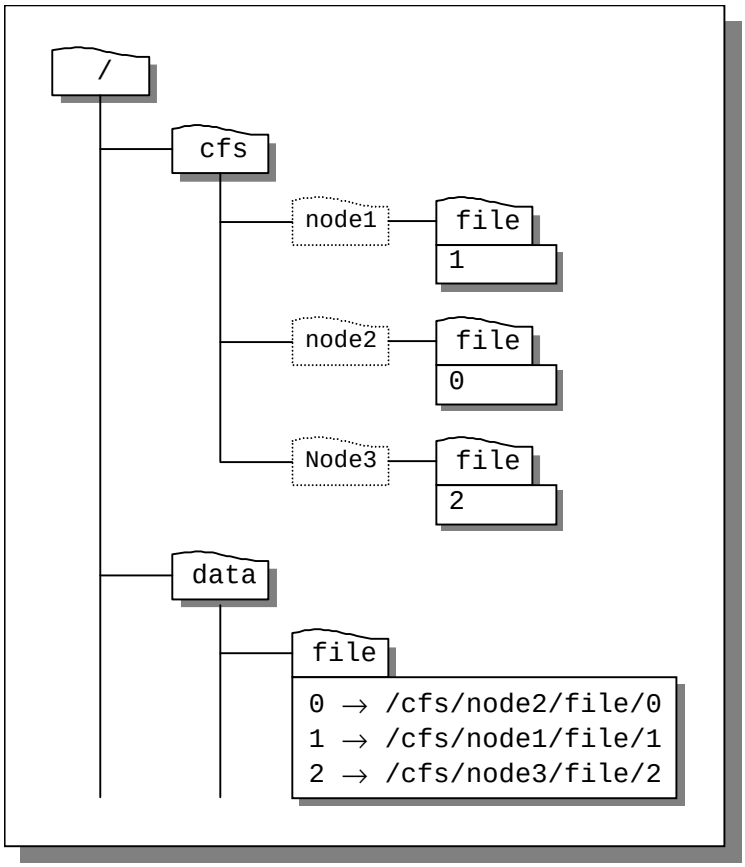

Рис. 6. Связь метаданных и субфайлов

Рис. 6 иллюстрирует состояние областей метаданных после проведения такой операции при открытии файла с именем file тремя параллельными ветвями одной задачи, работающими на трех различных узлах: node2 для ветви 0, node1 для ветви 1 и node 3 для ветви 2.

Аналогичным образом разрешаются ссылки и при открытии файлов на чтение. Следует, однако, заметить, что при большом объеме входных данных может оказаться выгодным введение дополнительного «нулевого» шага, на котором еще до запуска задачи осуществляется копирование необходимых файлов на узлы кластера, выделенные для счета данной задаче, с соответствующей корректировкой метаданных.

Отметим, что предложенная организация параллельной файловой системы позволяет использовать ее возможности и программам, не использующим MPI-IO. Обеспечивается это тем, что субфайлы представляют собой регулярные, последовательные файлы, доступ к которым возможен с использованием стандартных системных вызовов. Эта же особенность используется при реализации операций чтения/записи данных в ADIO.

Экспериментальная апробация модели хранения данных. Для оценки возможной эффективности предлагаемой организации файловой системы была проведена серия экспериментов на суперкомпьютере (СК) МВС-1000М, установленном в ГУ Межведомственный Суперкомпьютерный Центр (http://www.jscc.ru). Решающее поле МВС-1000М – 384 узловой кластер, содержащий 768 процессоров Alpha 21264A с тактовой частотой 667 МГц. Для доступа к файлам в системе используется гибридная сеть Fast Ethernet/Gigabit Ethernet: СК разбит на группы по 64 узла, в каждой из этих групп для коммуникаций используется сеть Fast Ethernet и многопортовый коммутатор. С помощью исходящих каналов эти коммутаторы объединяются в единую сеть высокопроизводительным Gigabit Ethernet коммутатором.

Контрольная задача была написана на языке Си с использованием библиотеки MPI. С помощью этой задачи оценивалась как скорость записи в файловую систему отдельными ветвями программ, так и агрегатная скорость записи.

Предлагаемая организация хранения данных учитывалась следующим образом: на всех узлах вычислительного кластера был активирован код поддержки NFS сервера и NFS клиента. На каждом из узлов была создана корневая вершина ФС /cfs и внутри нее запущен механизм автоматического монтирования таким образом, что при обращении к каталогам с символическими именами, совпадающими с именами узлов вычислительного кластера, происходило монтирование и дальнейшее использование NFS сервера указанного узла.

Входными данными задачи являлись параметры, задаваемые в командной строке и позволяющие варьировать следующие характеристики:

- общий размер записываемых данных;
- размер записываемого блока данных [таким образом выявлялась устойчивость системы при различной «агрессивности» клиентов, использующих как буферизованный так и не буферизованный ввод-вывод];
- смещение до «соседа» (этот параметр указывал удаление участвующих во взаимном тесте узлов в терминах его MPI номера) [таким образом выявлялось влияние топологии сети на эффективность обменов];

Для сравнительного анализа производительности был проведен аналогичный тест, в котором для физического хранения данных использовался NFS сервер, используемый в составе СК. В первом случае это был ПК, работающий под управлением ОС Linux, аналогичный узлу решающего поля, но имеющий в своем составе адаптер Gigabit Ethernet, а во втором – используемый в составе СК специализированный файл-сервер NetApp F840, соединенный с решающим полем двумя Gigabit Ethernet адаптерами. Рис. 7 и Рис. 8 отражают результаты проведенных измерений. Кривая NetApp соответствует результату, полученному для NetApp F840, кривая Linux – ОС Linux, остальные соответствуют различным вариантам запуска тестовой задачи для модели предлагаемой системы.

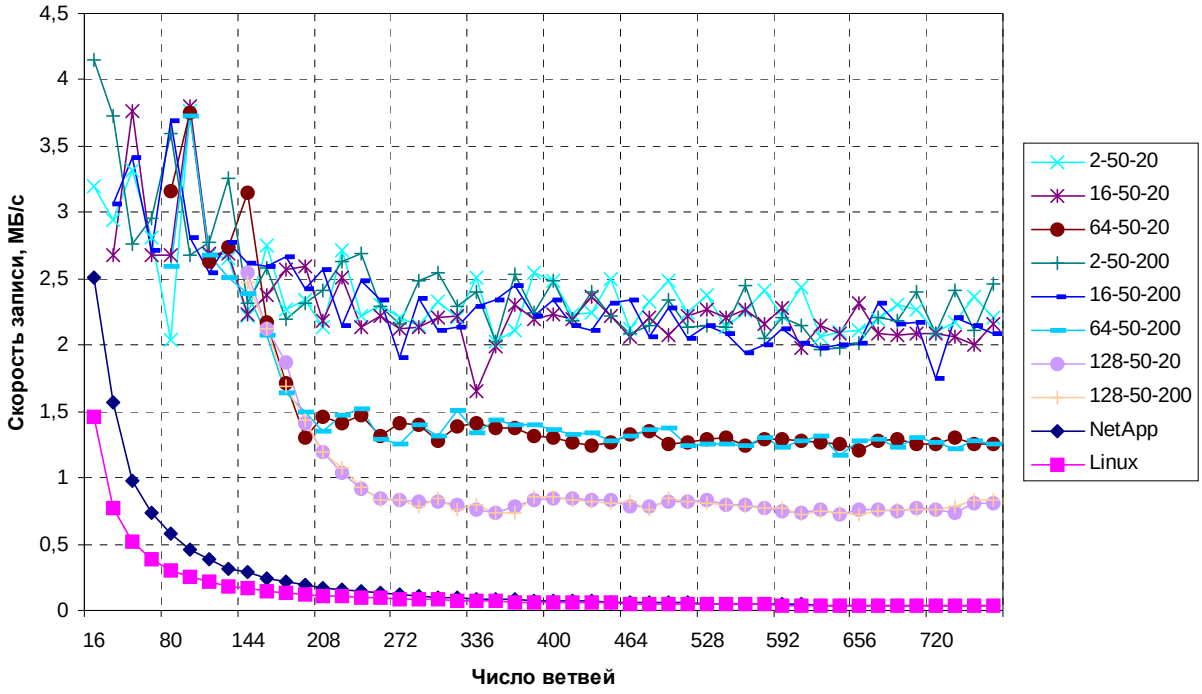

Рис. 7. Зависимость скорости записи от числа ветвей тестовой задачи на СК МВС-1000М

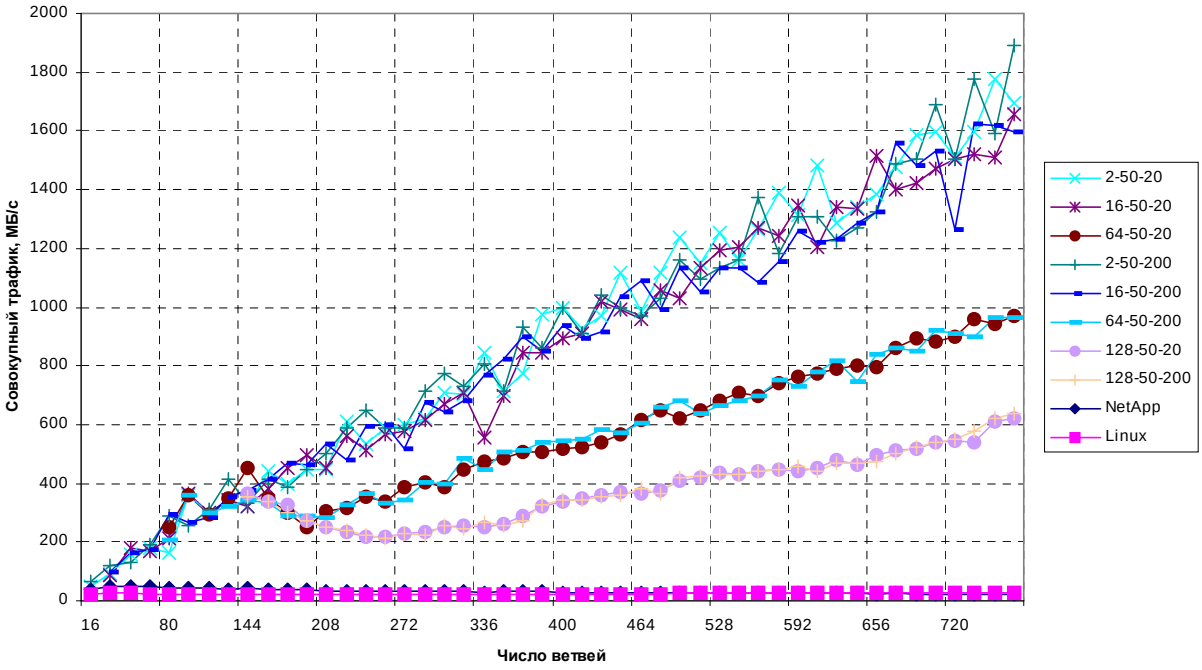

Рис. 8. Зависимость агрегатной скорости записи от числа ветвей тестовой задачи на СК МВС-1000М

Результаты измерений для традиционных сетевых файловых систем (с единственным выделенным файл-сервером) показывают, что их масштабируемость оказывается недостаточна для многопроцессорных кластеров. Скорость линейной записи в расчете на один узел деградирует на 2 порядка – при небольшом числе ветвей эта скорость достигает 2-3 мегабайта в секунду, а при увеличении их числа до 768 она падает до величин в 30

килобайт. Агрегатная скорость записи для NetApp F840 при этом падает с величины около 60 мегабайт в секунду до 23 мегабайт в секунду. Linux система при любом количестве ветвей показывает результат около 25 мегабайт в секунду.

Результаты, полученные для предлагаемой файловой системы, показывали практически линейную ее масштабируемость, при этом скорости обменов в основном определялись параметрами коммуникационной среды, т.е. сетями Fast/Gigabit Ethernet и используемыми коммутаторами. При этом в «установившемся» режиме скорость доступа в расчете на один узел изменялась не более чем на 10% и составляла в зависимости от расположения участвующих в тесте процессов от 1 до 3 мегабайт в секунду. Агрегатная скорость при этом линейно возрастала и достигала в «худшем» случае (когда все узлы вынуждены были обмениваться через сеть Gigabit Ethernet) 600 мегабайт в секунду (эта величина – максимальная пропускная способность коммутатора в данной топологии), и при «среднем» раскладе (когда обмениваются «соседи») доходила до 2 гигабайт в секунду.

Заключение

Предлагаемая схема организации хранения данных в ходе экспериментальной апробации продемонстрировала производительность, ограниченную главным образом возможностями коммуникационной среды, и хорошую масштабируемость. Таким образом, можно предположить, что реализация параллельной файловой системы с использованием рассмотренного подхода позволит получить характеристики эффективности на уровне существующих параллельных файловых систем.

Литература

1. *Message Passing Interface Forum.* MPI: A Message-Passing Interface Standard. Version 1.1, Июнь 1995.
2. *The MPI-IO Committee.* MPI-IO: A Parallel File I/O Interface for MPI, Version 0.5. World Wide Web http://lovelace.nas.nasa.gov/MPI-IO, Апрель 1996.
3. *P. F. Corbet, D. G. Feitelson.* The Vesta Parallel File System. ACM Transactions on Computer Systems, 14(3): 225-264 (1996)
4. *P. F. Corbet, D. G. Feitelson, J-P. Prost, G.S. Almasi, S.J. Baylor, A.S. Bolmarich, Y. Hsu, J. Satran, M. Snir, R. Colao, B.D. Herr, J. Kavaky, T.R. Morgan, and A. Zlotek.* Parallel file systems for the IBM SP computers. IBM Syst. J. 34(2), с. 222-248, 1995
5. *P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur.* PVFS: A Parallel File System For Linux Clusters. Proceedings of the 4th Annual Linux Showcase and Conference, Atlanta, GA, Октябрь 2000, с. 317-327
6. *W. Gropp, E. Lusk, N. Doss, and A. Skjellum.* A High Performance, Portable Implementation of the MPI Message-Passing Interface Standard. Parallel Computing, 22(6):789-828, Сентябрь 1996.
7. *R. Thakur, W. Gropp, E. Lusk.* An Abstract-Device Interface for Implementing Portable Parallel-I/O Interfaces. Proceedings Of the 6th Symposium on the Frontiers of Massively Parallel Computation, Октябрь 1996, с. 180-187.
8. *The Globus Project.* The Nexus Multithreaded Communication Library. World Wide Web http://www.globus.org/nexus/
9. *D. Culler, K. Keeton, L.T. Liu, A. Mainwaring, R. Martin, S. Rodrigues, K. Wright, C. Yoshikawa.* The Generic Active Message Interface Specification. White Paper, 1994 (World Wide Web http://now.cs.berkeley.edu/Papers/Papers/gam_spec.ps)