УДК 004

ОПТИМИЗАЦИЯ ОБРАБОТКИ БОЛЬШИХ МАССИВОВ ДАННЫХ В КЛАСТЕРНЫХ СИСТЕМАХ

Е.В. Назаренко, В.Г. Тульчинский, П.Г. Тульчинский

Институт кибернетики имени В.М. Глушкова НАН Украины,
03687, Киев, проспект Академика Глушкова, 40, т. 526 3603,
pgt@ukr.net

Для задач параллельной обработки больших наборов данных узким местом, как правило, оказывается файловое хранилище. Рассмотрено влияние упаковки данных на скорость вычислений. На основе ранее разработанной модели получены оценки минимального времени выполнения программ с учетом времени упаковки/распаковки и коэффициента сжатия данных. Полученные оценки опробованы на примере сжатия временных кубов для ускорения процедуры миграции сейсмограмм.

For tasks of massive dataset processing the file storage usually provided to be a bottleneck. The data compression affect on computation speed is examined. On the base of the earlier built model the minimal execution time estimates with accounting the pack/unpack time and the data compression coefficient are obtained. The obtained estimates have been verified on the sample of time cubes compression for acceleration of seismic migration procedure.

Введение

Значительную часть научных и прикладных задач, решаемых на вычислительных кластерах, составляют задачи однородной обработки больших массивов данных. Они, как правило, легко распараллеливаются методом геометрической декомпозиции, так как обладают естественной параллельностью по данным. Такие задачи возникают, в частности, в геофизике (обработка результатов сейсмических и электромагнитных исследований земной коры), в ядерной физике, метеорологии, биологии, при обработке видео, естественно-языковых текстов, в экономических расчетах (OLAP, Data Mining и статистическая обработка больших баз данных) [1]. В связи с относительной простотой масштабирования вычислительных ресурсов в кластерной архитектуре за последние годы достигнут значительный рост числа процессорных элементов, доступных в современных кластерах. Однако возможности наращивания производительности систем хранения данных отстают от быстро растущей скорости вычислений. Для решения задач массивной обработки данных узким местом, как правило, становится система хранения. Дело в том, что для эффективного обслуживания параллельных запросов необходимо почти линейное относительно числа узлов увеличение скорости ввода-вывода, в то время, как рост производительности аппаратуры хранения данных отстает от роста скорости процессоров [2].

Влияние файловых операций на общую производительность параллельной программы исследовались еще в конце 60-х, в частности, одним из родоначальников теории параллельных вычислений Амдалом [3]. И хотя впоследствии производительность ввода-вывода ненадолго оказалась вне фокуса исследований высокопроизводительных вычислений [4, 5], так как размещение всех данных в оперативной памяти считалось обязательным условием эффективности вычислений, эта тема продолжала разрабатываться в связи с сетевыми файловыми системами [6 – 8].

В середине 90-х появление вычислительных кластеров вызвало широкий интерес к программным и аппаратным архитектурам, способным поддержать высокую степень распараллеливания при вводе-выводе данных. Уже в ранних работах [9, 10] была обнаружена главная специфика доступа к данным параллельных программ с интенсивным вводом-выводом: непоследовательное чтение небольших порций данных. Для оптимизации работы системы хранения в таком экстремальном режиме были разработаны специальные методы кэширования [11, 12]. Кэширование реализовывалось в форме параллельных файловых систем и параллельных библиотек ввода-вывода. Параллельные файловые системы оптимизируют использование аппаратуры хранения данных [13 – 17], а параллельные библиотеки ввода-вывода – управление доступом к данным внутри самой программы [18 – 25]. В [26] предложена параллельная система хранения данных для грид-систем.

В [27 – 29] проблема эффективной параллельной обработки больших объемов данных рассматривалась не с системной, а с прикладной точки зрения. А именно:

- оптимизации распределения ресурсов существующего кластера, т. е. подбор числа процессоров и способа размещения данных, обеспечивающих, минимальное время выполнения программы,
- оценки ожидаемого эффекта от распараллеливания последовательной программы для использовании на существующем кластере,
- оптимизации архитектуры кластера и аппаратно-программного решения под определенный класс задач.

Продолжая тему оптимизации параллельной обработки больших объемов данных, в этой работе мы рассматриваем применение алгоритмов сжатия данных для снижения нагрузки на файловую систему.

Эффективность упаковки данных при интенсивном вводе-выводе

В [28] предложена следующая модель синхронной параллельной обработки большого объема сейсмических данных на кластере, на котором N процессов выполняются параллельно. Работа каждого процесса может быть представлена в виде чередования расчетов продолжительностью t_i^n и обращений к данным – чтения или записи файлов – продолжительностью τ_i^n с ожиданием продолжительностью ω_i^n. Еще S процессов или устройств обслуживают обращения к данным. Каждый из них ждет поступления запроса от одного из вычислительных процессов, затем выполняет его. Пока обслуживающий процесс выполняет запрос, на него могут поступить другие запросы. Все они ставятся в очередь неограниченной емкости и обслуживаются в порядке поступления. Коммуникационные расходы, не относящиеся к чтению/записи данных, предполагаем пренебрежимо малыми. Время завершения всех N процессов, состоящих из i шагов, обозначим

$$T_N = \max_{n=1..N}\left(t_0^n + \omega_1^n + \tau_1^n + t_1^n + \omega_2^n + \tau_2^n + t_2^n + ... + \omega_i^n + \tau_2^n + t_2^n\right).$$

Пусть общие затраты на чтение/запись данных последовательной программы $D_1 = \sum \tau_i$, в том числе на операции чтения приходится R_1. Обозначим суммарное время выполнения обязательных для каждого процесса (не распараллеливаемых) операций чтения τ_{sec}^R, записи – τ_{sec}^W, максимальное время одной операции чтения/записи – $\tau_{\max}$, среднее время операции доступа к данным – $\mu(\tau)$, средний квадрат – $\mu(\tau^2)$. Все перечисленные параметры можно непосредственно измерить. Обозначим долю распараллеливаемой части расчетов $0 < \eta \le 1$. η определяется путем небольшого числа экспериментов с параллельной программой. Доля расчетной части последовательной программы составляет $\theta = (T_1 - D_1)/T_1$, доля чтения – $\rho = R_1/D_1$.

Поведение такой модели зависит от соотношения между временем расчетов и временем доступа к данным. Будем рассматривать два предельных случая: со слабой загрузкой, когда очередь запросов на доступ к данным одной операции успевает «рассосаться» до начала следующей операции доступа к данным ($\forall i \sum_n \tau_i^n << t_{i+1}^n$), и с полной загрузкой, когда эта очередь, возникнув на первом шаге, продолжает расти

$$(\forall i \sum_n \tau_i^n \ge t_{i+1}^n).$$

При *расчленении* непересекающиеся подмножества данных управляются процессами на разных узлах. В случае расчленения время выполнения зависит от распределения данных между фрагментами, но дублирование записи не требуется. Для случайного распределения можно оценить среднее значение времени работы при слабой загрузке как [29]:

$$\mu(T_N) = T_1\left(\frac{1}{2}C_N + D_{NS} + \sqrt{\frac{1}{4}C_N^2 + D_{NS}^2\frac{\mu(\tau^2)}{2\mu(\tau)^2}}\right), \quad (1)$$

где $C_N = \theta\left(1 - \eta\frac{N-1}{N}\right)$, $D_{NS} = \frac{1}{NS}\left(1 - \theta + (N-1)\frac{\tau_{\text{sec}}^W + \tau_{\text{sec}}^R}{T_1}\right)$. При полной загрузке:

$$\mu(T_N) = \frac{1-\theta}{S}T_1 + (N-1)\frac{\tau_{\text{sec}}^W + \tau_{\text{sec}}^R}{S} + t_{const}, \text{ где } t_{const} = t_0^1 + t_i^N. \quad (2)$$

Отметим, что $c_N = C_N T_1$ – это время «чистых» расчетов одного из N процессов, а $d_N = D_{NS}T_1$ – это «чистое» время чтения-записи данных такого процесса. Обе величины нетрудно измерить. Перепишем (1) и (2), выразив через новые переменные:

$$\mu(T_N) = \frac{1}{2}c_N + d_N + \sqrt{\frac{1}{4}c_N^2 + d_N^2\frac{\mu(\tau^2)}{2\mu(\tau)^2}} \quad \text{при слабой загрузке,} \quad (3)$$

$$\mu(T_N) = Nd_N + t_{const} \quad \text{при полной загрузке.} \quad (4)$$

Результаты опробования [28, 29] показали, что точка перегиба экспериментального графика зависимости времени выполнения программы массивной параллельной обработки данных находится вблизи точки пересечения графиков слабой и полной загрузки (см., например, рис. 1).

В этой точке перегиба достигается максимальная для данной программы производительность, соответствующая среднему времени выполнения $t_N = \mu(T_N): \forall n \ne N\ \mu(T_n) \ge \mu(T_N)$. При уменьшении объема операций чтения-записи файлов за счет сжатия данных время выполнения программы уменьшается в результате одновременного воздействия двух факторов: уменьшения d_N и увеличения оптимального числа процессов N. В то же время, увеличение объема чистых расчетов c_N в связи с затратами на упаковку-распаковку данных ограничивает эффект от сжатия данных.

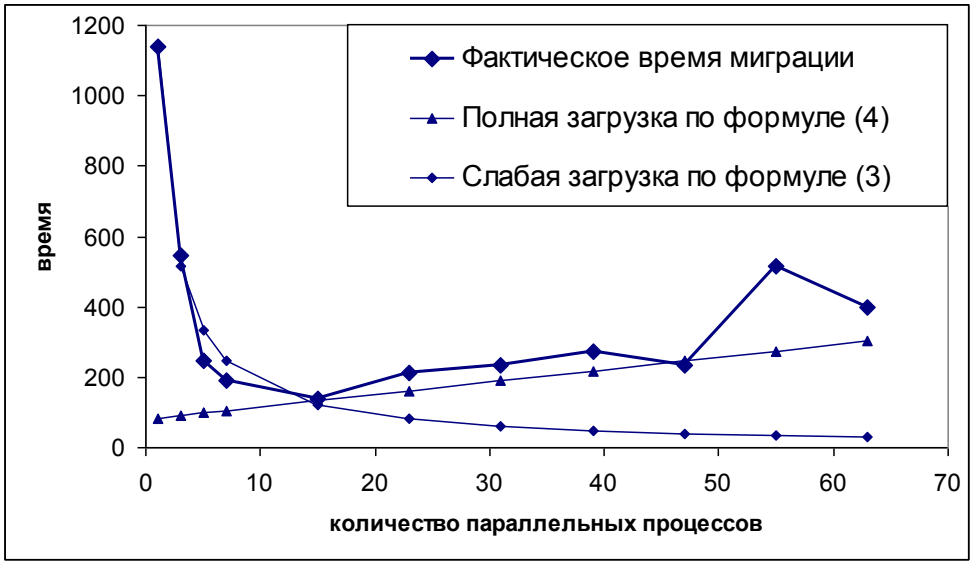

Рис. 1. Глубинная пре-стек миграция Кирхгофа в режиме расчленения данных (параллельная файловая система на основе винчестеров 8 узлов)

Построим соотношение изоэффективности сжатия, показывающее, насколько временные затраты связанные со сжатием информации компенсируются уменьшением затрат на чтение-запись в связи с сокращением объема данных.

Подставив (4) в (3) для точки пересечения получаем:

$$Nd_N + t_{const} \approx \frac{1}{2}c_N + d_N + \sqrt{\frac{1}{4}c_N^2 + d_N^2 \frac{\mu(\tau^2)}{2\mu(\tau)^2}},$$

откуда путем несложных преобразований:

$$c_N \approx (N-1)d_N + t_{const} - \frac{d_N}{N-1+(t_{const}/d_N)} \frac{\mu(\tau^2)}{2\mu(\tau)^2}, \quad (5)$$

Поскольку нас интересует только случай сильного распараллеливания при значительном времени работы программы, целесообразно предположить $t_{\text{const}} << d_N$ и $N >> 1$. В этом случае (5) можно упростить до:

$$c_N \approx (N-1)d_N. \quad (6)$$

Соотношение (6) действительно только в точке перегиба графика времени выполнения.

При *дублировании* на всех или нескольких узлах расположены точные копии данных. В частном случае *централизации* единственная копия данных управляется одним процессом на одном узле кластера или в дисковой стойке ($S=1$).Для стратегии дублирования при слабой загрузке время работы параллельной программы составляет [29]:

$$\mu(T_N) = \left(\theta\left(1-\eta\frac{N-1}{N}\right) + \frac{1-\theta}{N}\left(1-\rho\frac{S-1}{S}\right)\right)T_1 + \frac{N-1}{N}\left(\tau_{\text{sec}}^W + \frac{\tau_{\text{sec}}^R}{S}\right) + \left\lceil\frac{N-1}{S}\right\rceil\tau_{\max}. \quad (7)$$

При большом числе процессов и интенсивном обращении к файлам обслуживание чтения и записи данных почти полностью определяет время работы:

$$\mu(T_N) = (1-\theta)\left(1-\rho\frac{S-1}{S}\right)T_1 + (N-1)\left(\tau_{\text{sec}}^W + \frac{\tau_{\text{sec}}^R}{S}\right) + t_{const}. \quad (8)$$

В случае дублирования время «чистых» расчетов одного из N процессов то же, а время «чистого» чтения-записи данных отличается:

$$d_N = \frac{1-\theta}{N}\left(1-\rho\frac{S-1}{S}\right)T_1 + \frac{N-1}{N}\left(\tau_{\text{sec}}^W + \frac{\tau_{\text{sec}}^R}{S}\right). \quad (9)$$

Обозначим максимальное измеренное время выполнения одной файловой операции параллельной программой

$$\tau_{\max}^S = \frac{1}{S}\tau_{\max}. \quad (10)$$

Перепишем (7) и (8), выразив через новые переменные (9) и (10):

$$\mu(T_N) = c_N + d_N + (N-1)\tau_{\max}^S \text{ при слабой загрузке,} \quad (11)$$

$$\mu(T_N) = Nd_N + t_{const} \text{ при полной загрузке.} \quad (12)$$

Откуда для точки пересечения графиков (11) и (12) получаем:

$$c_N \approx (N-1)\left(d_N - \tau^S_{\max}\right) + t_{const},$$

Из предположения долгого времени работы параллельной программы $d_N >> \tau^S_{\max}$:

$$c_N \approx (N-1) d_N .$$

Таким образом, независимо от стратегии распределения данных при оптимальном числе процессоров соотношение расчетов и операций доступа к данным одного процесса описывается формулой (6).

Оценим влияние в результата применения алгоритма сжатия, который уменьшает объем данных, а значит и чистое время ввода-вывода, косвенно способствуя увеличению оптимального числа процессоров. Пусть коэффициент сжатия данных $k>1$, z_N – дополнительное время упаковки и распаковки данных, ввод-вывод которых при реализации алгоритма без сжатия – d_N, а относительная скорость распаковки $v = z_N / d_N$.

Рассмотрим условие, при котором программа со сжатием данных работает на N^* процессорах быстрее, чем исходная программа на N процессорах:

$$z_N \frac{N}{N^*} - c_N\left(1 - \frac{N}{N^*}\right) < d_N\left(1 - \frac{1}{k}\right). \tag{13}$$

Путем несложных преобразований из (13) получается:

$$\frac{N}{N^*} < E = \frac{\frac{k-1}{k} + \frac{c_N}{d_N}}{v + \frac{c_N}{d_N}}. \tag{14}$$

Назовем E изоэффективностью сжатия данных. В случае, если N – оптимальное число процессоров для исходной программы, согласно (6) выражение для E можно переписать как:

$$E = \frac{N - \frac{1}{k}}{N + r - 1}. \tag{15}$$

Если $E>1$, сжатие эффективно уже для $N^* = N$, т. е. обеспечивает ускорение без увеличения числа процессоров. Это – самый лучший вариант.

Если максимально доступное число процессоров кластера – $N_{\max}$ и $E < N/N_{\max}$, на этом кластере рассмотренный алгоритм сжатия не даст ускорения. Это – самый плохой вариант.

В остальных случаях сжатие ускоряет расчет при условии увеличения числа задействованных процессоров.

В целом изоэффективность позволяет характеризовать ожидаемый эффект ускорения параллельной программы от применения некоторого метода сжатия данных на основе анализа ее текущей производительности и обычных характеристик алгоритма сжатия – коэффициента сжатия и времени работы.

Применение сжатия при миграции сейсмических данных

Сейсмические исследования – основной поверхностный метод зондирования земных недр в поисках месторождений нефти и газа. Кроме того сейсмические исследования выполняются в связи с изучением землетрясений и в инженерной геологии. Данные сейсмических исследований современными методами могут достигать 100 терабайт на один объект [30]. Сжатие сейсмических данных традиционно применяется в связи с их хранением и передачей по сети. Разработаны методы сжатия сейсмических данных с потерями (прежде всего на основе вейвлет преобразования и его модификаций, а также адаптивных локальных дискретных косинус и синус преобразованиях, преобразовании Уолша, линейных преобразованиях с перекрытиями) и без потерь, т. е. с точным восстановлением исходных данных (на основе битовых масок, предсказывающего кодирования, модификаций LZW), а также двухэтапные процедуры, состоящие из преобразования с потерями и обработки остатков [31].

Однако при исследовании производительности пре-стек миграций Кирхгофа 3D мы обнаружили, что параллельная файловая система обеспечивает достаточно эффективное чтение сейсмограмм. Ее производительности недостаточно на этапе суммирования, когда для обработки одной трассы необходимо использовать два набора времен: от источника и от приемника. При регулярной схеме наблюдений разные координаты источников и приемников измеряются тысячами, а число трасс – миллионами. Поскольку загрузка готового файла времен занимает на нашем кластере значительно меньше времени, чем его расчет, один файл времен приходится подгружать вместе с трассой. Типичный объем файла времен – десятки МБ, что создает предпосылки для применения сжатия.

Времена прихода падающей волны плавно возрастают в стороны от источника или приемника, с которым связан файл. Эта форма сильно отличается от типичной волновой формы сейсмических сигналов, поэтому для сжатия мы пробовали другие методы: упаковка приращения (УП), линейные альтернансные сплайны Чебышева (ЛАС) [32], наилучшая Чебышевская аппроксимация полиномами (НЧА) [33] и их сочетание с LZW (zip). Выбор методов определился соображениями гарантированной точности: времена используются для переноса сигнала с трассы, поэтому можно использовать алгоритм сжатия с потерями, однако он должен гарантировать выбор той же точки трассы, что и при использовании несжатых данных. Как правило, для этого достаточна точность аппроксимации 1–2 мс. Условию минимизации максимального отклонения отвечают методы НЧА и ЛАС. УП – простой эвристический метод. Его идея заключается в том, что при нормальной пространственной густоте сетки (до сотни метров) и типичных скоростях (от тысячи м/с) приращение времени прихода волны между соседними узлами не превысит 128 мс. УП хранит в узлах не значения времен, а их приращения в мс в виде коротких целых чисел (один байт на точку). УП – базовый формат хранения времен в нашей программе миграции обеспечивает сжатие в 4 раза и пренебрежимо быструю упаковку-распаковку (одно сложение на точку). Остальные методы сравниваются с УП.

Исходные данные для расчета получены путем экспериментов на кластере Инпарком [34]. В оптимальном режиме миграция выполнялась на 15 узлах. Для простоты мы ограничились собственно миграцией, предполагая, что расчет временных кубов уже выполнен. (Запись временных кубов на этапе расчетов выполнялась в 520 раз реже, чем чтение на этапе миграции). Для тестирования выбранных алгоритмов сжатия мы использовали 10 случайно выбранных файлов времен. Среднее время чтения всех десяти кубов данных составило 26 с. Остальные данные сведены в таблицу.

Таблица. Сравнение алгоритмов сжатия данных

Алгоритм сжатия	Время упаковки 10 файлов (с)	Коэффициент сжатия r	Время распаковки 10 файлов (с)	Изоэффектив-ность сжатия E
LZW	112	7,0	30	0,980
ЛАС	185	2,8	5	1,032
ЛАС +LZW	224	10,7	9	1,039
НЧА	1320	1,9	17	0,988
НЧА+LZW	1370	2,9	29	0,970

Два алгоритма сжатия получили оценку изоэффективности сжатия $E > 1$, которая означает наилучшую рекомендацию для применения. ЛАС, второй по изоэффективности, но более быстрый и простой в реализации, был выбран для реализации в программе миграции. В результате его применения было достигнуто реальное ускорение: 146 мин. счета вместо 171 мин. на том же наборе данных.

Заключение

В большом числе научных и технических приложений необходимо обрабатывать большие объемы данных. Несмотря на успехи в развитии аппаратуры хранения данных, параллельных файловых систем и параллельных библиотек ввода-вывода, система хранения данных часто оказывается «узким местом» при решении таких задач на вычислительных кластерах с большим числом узлов. В таких случаях время расчета не только перестает уменьшаться, но и начинает расти с увеличением числа используемых параллельных процессорных элементов. Применение сжатия данных может существенно снизить нагрузку на систему хранения данных. В результате максимально достижимое ускорение повышается не только из-за снижения «чистого» времени ввода-вывода, но и в связи с продвижением точки перегиба графика производительности в сторону большей степени распараллеливания. Благодаря этому двойному эффекту сжатие данных при параллельных вычислениях может оказаться выгодным даже тогда, когда упаковка-распаковка требуют больших вычислений.

Полученные оценки позволяют оценить целесообразность внедрения того или иного алгоритма упаковки данных для оптимизации выполнения программы на кластере. Их использование в сочетании с формулами прогнозирования производительности программы позволяет находить наилучший баланс между скоростью, объемом данных и ошибкой восстановления.

1. *Oldfield R., Kotz D.* Scientific applications using parallel I/O / High Performance Mass Storage and Parallel I/O: Technologies and Applications. – New York: IEEE Computer Society Press and John Wiley & Sons. – 2001. – P. 655–666.
2. *Hennessy J.L., Patterson D.A.* Computer architecture: A quantitative approach. – Boston: Morgan Kaufmann Publishers. – 2006. – 696 p.
3. *Amdahl G.M.* Validity of the single-processor approach to achieving large scale computing capabilities // Proc. AFIPS Spring Joint Computer Conf. – Atlantic City: AFIPS Press. – 1967. – P. 483–485.
4. *Kumar V., Gupta A.* Analyzing Scalability of Parallel Algorithms and Architectures // J. of Parallel and Distributed Computing. – 1994. – Vol. 22. – P. 379–391.
5. *Глушков В.М., Капитонова Ю.В., Летичевский А.А.* К теории проектирования схеммного и программного оборудования микропроцессорных ЭВМ // Кибернетика. – 1978. – № 6. – С. 1–15.
6. *Bell J., Casasent D., Bell C.G.* An Investigation of Alternative Cache Organizations // IEEE Transactions on Computers. – 1974. – Vol. C-23 (4). – P. 346–351.
7. *Kung H. T.* Memory requirements for balanced computer architectures // Proceedings of the IEEE Symp. on Computer Architecture. – IEEE Press. – 1986. – P. 49–54.
8. *Nitzberg B.* Performance of the iPSC/860 Concurrent File System / Technical Report RND-92-020 of NAS Systems Division. – NASA Ames. – 1992.
9. *Crandall P. E., Aydt R. A., Chien A. A., Reed. D. A.* Input/output characteristics of scalable parallel applications // Proc. of Supercomputing ’95 Proc. of Supercomputing '95. – San Diego: IEEE Computer Society Press. – 1995.
10. *Nieuwejaar N., Kotz D., Purakayastha A., Ellis C. S., Best M.* File-access characteristics of parallel scientific workloads // IEEE Transactions on Parallel and Distributed Systems. – 1996. – No 7(10). – P. 1075–1089.
11. *Фальфушинский В.В.* Кэширование в кластерных системах // Компьютерная математика. – 2008. – №2. – С. 64–73.
12. *Nitzberg W. J.* Collective Parallel I/O. (PhD thesis). – Oregon: University of Oregon. – 1995.
13. *Corbett P. F., D. G. Feitelson.* The Vesta parallel file system / High Performance Mass Storage and Parallel I/O: Technologies and Applications. - New York: IEEE Computer Society Press and Wiley. – 2001. – P. 285–308.
14. *Moyer S. A., Sunderam V. S.* PIOUS: a scalable parallel I/O system for distributed computing environments // Proc. of Scalable High-Performance Computing Conf. – New York: IEEE Computer Society Press. – 1994. – P. 71–78.
15. *Nieuwejaar N., Kotz D.* The Galley parallel file system // Parallel Computing. – No 23(4). – 1997. – P. 447–476.
16. *Schmuck R. L. F. B.* GPFS: A shared-disk file system for large computing clusters // Proc. of 5th Conf. on File and Storage Technologies. – San Jose: IBM Almaden Research Center. – 2002. (www.usenix.org/events/fast02/full_papers/schmuck/schmuck.pdf).
17. *Schwan P.* Lustre : Building a file system for 1,000-node clusters. // ACM Transactions on Computer Systems (TOCS). – 2003. – Vol. 14(2). – P.200-222. (http://www.kernel.org/doc/ols/2003/ols2003-pages-380-386.pdf)
18. *Lebre A., Huard G., Denneulin Y.* I/O Scheduling Service for Multi-Application Clusters // IEEE Int. Conf. on Volume. – Barcelona: IEEE Computer Society Press. – 2006. – P. 21–23.
19. *Kotz D.* Disk-directed I/O for MIMD multiprocessors. // Proc. of the 1994 Symp. on Operating Systems Design and Implementation. – Dartmouth: IEEE Computer Society Press. – 1994. – P. 61–74.
20. *Seamons K. E., Chen Y., Jones P., Jozwiak J., Winslett M.* Server-directed collective I/O in Panda. // Proc. of Supercomputing '95. – San Diego: IEEE Computer Society Press. – 1995.
21. *Bennett R., Bryant K., Sussman A., Das R., Saltz J.* Jovian: A framework for optimizing parallel I/O. // Proc. of Scalable Parallel Libraries Conf. - Mississippi State: IEEE Computer Society Press. – 1994. – P.10–20.
22. http://www.mcs.anl.gov/research/projects/romio/
23. http://www.nersc.gov/nusers/resources/software/libs/io/mpiio.php
24. http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf
25. http://aioli.imag.fr/
26. *Oldfield R., Kotz D.* Armada: A parallel I/O framework for computational grids // Future Generation Computing Systems (FGCS). – 2002. – Vol. 18(4). – P. 501–523.
27. *Гречко В.О., Гудзенко В.В.* Продуктивность параллельной СУБД в кластерных системах // Компьютерная математика. – 2005. – №3. – С. 71-79.
28. *Тульчинский В.Г., Чарута А.К.* Оценка времени обработки данных в кластерных системах // Проблеми програмування. – 2006. – №2-3. – С. 118-123.
29. *Перевозчикова О.Л., Тульчинский В.Г., Ющенко Р.А.* Построение и оптимизация параллельных компьютеров для обработки больших объемов данных // Кибернетика и системный анализ. – 2006. – №4. – С. 117–129.
30. *Donoho P.L.* Seismic data compression: Improved data management for acquisition, transmission, storage, and processing. – Proc. Seismic. – 1998.
31. *Wang W., Mishra P.* A Partitioned Bitmask-based Technique for Lossless Seismic Data Compression. CISE Technical Report # 08-452. – University of Florida. – 2008. – 10 p. (http://www.cise.ufl.edu/~prabhat/Publications/seismicTR.pdf)
32. *Исаев В.К., Плотников СА.* Обратная задача Чебышева и сплайны Чебышева // Труды Математического института РАН. – 1995. – Т.221. – С. 164-185.
33. *Каленчук-Порханова А.А., Вакал Л.П.* Наилучшая чебышевская аппроксимация для сжатия численной информации // Компьютерная математика. – 2009. – № 1. – С. 99–107.
34. http://www.inparcom.com/