УДК 004.67

П.Ю. Добрачев
Московский физико-технический институт (государственный университет)

Подсистема «Динамика» для хранения и статистического анализа информации о потреблении вычислительных ресурсов программы «Университетский кластер»

Подсистема «Динамика» предназначена для структурирования, хранения и анализа информации о потреблении ресурсов пользователями системы. Данная подсистема написана на языках Python и PL/pgSQL и использует СУБД PostgreSQL для хранения информации. Программное обеспечение анализирует журнальный файл, который ведёт сервер очередей qserver, а затем собранная информация структурируется и сохраняется в базе данных. Подсистема содержит различные приложения, которые собирают разнообразную статистику: статистика по коду завершения задачи, среднее время ожидания в зависимости от числа запрошенных процессоров, статистика по прерванным задачам в зависимости от времени обработки, среднее время ожидания в зависимости от типа задачи, статистика по количеству задействованных процессоров.

Ключевые слова: подсистема «Динамика», статистический анализ, компьютерные ресурсы, журнальный файл, сервер очередей, база данных.

Межведомственный Суперкомпьютерный Центр РАН (МСЦ) является крупнейшим и самым мощным суперкомпьютерным центром в сфере науки и образования, в десять раз превышающим по производительности другие крупные вычислительные центры России. Огромное количество самого современного оборудования организовано в сложную вычислительную систему, которая даёт возможность получить максимально возможную производительность.

I. Цель создания подсистемы «Динамика»

В связи со сложной организацией МСЦ встает задача мониторинга производительности всей системы в целом. Необходимо отслеживать эффективность работы нескольких кластеров, которые входят в состав вычислительного центра. Также следует отслеживать потребление ресурсов различными пользователями в соответствии с выделенными для них квотами, а также квотами для различных групп и научных направлений. Все это необходимо для повышения общей эффективности использования ресурсов суперкомпьютерного центра. Поэтому была поставлена задача по созданию подсистемы «Динамика» для хранения и статистического анализа информации о потреблении вычислительных ресурсов программы «Университетский кластер».

Подсистема «Динамика» разработана на базе СУБД PostgreSQL с использованием высокоуровневого языка программирования Python и процедурного языка PL/pgSQL [1]. В данную подсистему входят следующие задачи.

1. Хранение информации о ходе работы различных суперкомпьютеров, которые входят в состав МСЦ.

2. Хранение информации о потреблении задачами или пользователями вычислительных ресурсов, таких, как процессорное время и оперативная память.

3. Статистический анализ информации о потреблении вычислительных ресурсов с целью контроля эффективности работы системы.

4. Осуществление быстрого поиска любого события и любой информации, которые были отражены в журнальных файлах различных кластеров.

Подсистема «Динамика» даст возможность анализировать эффективность рабо-

ты компонентов системы за различные периоды времени. Статистическая обработка информации, собранной из журнальных файлов, позволит оценивать эффективность изменений, которые вносятся по мере работы суперкомпьютеров. Мониторинг различных параметров производительности суперкомпьютеров МСЦ сможет выявить системные сбои и ошибки на самых ранних этапах, что позволит избежать потери производительности.

II. Структура подсистемы

Структура подсистемы «Динамика» состоит из базы данных «Динамика» на основе СУБД PostgreSQL [2] и программ (скриптов), написанных на языках Python и PL/pgSQL. Источником информации для статистического анализа выступают журнальные файлы, которые ведутся сервером очередей qserver. Для того чтобы переписать информацию, собранную в журнальном файле, в базу данных «Динамика», была написана специальная программа на языке Python, которая разбирает каждую строчку в журнальном файле и записывает все разобранные параметры в соответствующие ячейки базы данных. После того как интересующая информация занесена в БД «Динамика», появляется возможность провести статистический анализ, используя специально созданные для этого программы, которые для работы с базой данных используют процедурный язык PL/pgSQL.

В качестве СУБД для создания базы данных «Динамика» была выбрана PostgreSQL, так как именно PostgreSQL считается одной из самых совершенных СУБД, распространяемых на условиях открытых исходных текстов [3]. В PostgreSQL реализованы многие возможности, обычно присутствующие только в коммерческих СУБД, таких, как Oracle [4]. Далее перечислены основные возможности PostgreSQL версии 8.2.x: в основе СУБД лежит объектно-реляционная модель, есть полноценная поддержка SQL, поддерживается проверка целостности ссылок, поддержка различных API (в том числе Python), наличие процедурного языка PL/pgSQL [5], наличие технологии MVCC (Multi-Version Concurrency Control) для предотвращения лишних блокировок.

При создании программ для статистического анализа информации, которая хранится в базе данных «Динамика», был использован процедурный язык PL/pgSQL [6]. Язык PL/pgSQL позволяет группировать на сервере код SQL и программные команды, что приводит к снижению затрат сетевых и коммуникационных ресурсов, обусловленных частыми запросами данных со стороны клиентских приложений и выполнением логической обработки этих данных на удалённых хостах. По сути, при работе с СУБД PostgreSQL процедурный язык PL/pgSQL используется в тех случаях, когда уже невозможно выбрать и обработать интересующие данные обыкновенными SQL-запросами [7].

Благодаря различным API, которые поддерживает СУБД PostgreSQL, в качестве языка для написания программы для разбора журнального файла был выбран высокоуровневый язык программирования Python [8]. Поскольку достижение высокой производительности для такой программы не являлось основной целью, был выбран язык, который позволяет быстро создавать подобные скрипты разработчику, а также имеет легко читаемый код. Для того чтобы программа, написанная на языке Python, могла подсоединяться к СУБД PostgreSQL, был использован специально предназначенный для этого модуль PyGreSQL-3.8.

III. Журнал событий сервера очередей

Сервер очередей qserver ведёт очередь параллельных задач согласно принципам планирования очередей. Также сервер очередей qserver ведёт протокол своей работы в журнальном файле qserv.log. Именно в этот журнальный файл записывается вся собранная с суперкомпьютера информация о работе системы. Файл qserv.log является единственным источником информации, который используется при работе подсистемы «Динамика». Для того чтобы переписать информацию из журнального файла в базу данных «Динамика», была создана специальная программа qserv-stat.py. В файл qserv.log записы-

вается большое количество различных параметров о состоянии системы, что даёт широкие возможности при сборе статистической информации.

Файл журнала событий (лог-файл) состоит из строк, каждая строка — одно событие. Общий формат строки следующий:

<тип> <дата время> <информация>

Первым символом строки идёт тип события, который может принимать следующие значения:

— '*' — информационное сообщение;

— '+' — сообщение о движении задач в очереди;

— '-' — сообщение об ошибке;

— '! ' — важное системное сообщение.

Рассмотрим основные события, заносимые в журнал (строки из журнального файла приведены в качестве примера).

1. ! 23.10.00 08:57: start free: 10 total: 15 — сообщение о старте сервера очередей с указанием общего числа (total) доступных процессоров и числа свободных процессоров на момент старта (free).

2. +23.10.00 11:57: queue user: kugush task: gen.7 CPU: 4 time: 6000 quant: 300 — сообщение о постановке в очередь задачи (task) gen.7 пользователя (user) kugush. Задача требует 4 процессора (CPU) на 6000 минут (time). Задача является фоновой с квантом (quant) 300 минут.

3. +23.10.00 13:43: run user: kugush task: gen.7 CPU: 4 time: 6000 quant: 300 pid: 559 — сообщение о запуске задачи (task) gen.7 пользователя (user) kugush. Задача требует 4 процессора (CPU) на 6000 минут (time). Задача является фоновой с квантом (quant) 300 минут.

Необходимо отметить, что число параметров о состоянии системы могло бы быть значительно больше, однако это неминуемо повлечет за собой снижение производительности системы.

IV. Программа qserv-stat.py

Для того чтобы записать журнальную информацию из файла qserv.log в базу данных «Динамика», была написана программа qserv-stat.py на языке Python. Данная программа разбирает каждую строку журнального файла и записывает каждый параметр в соответствующую ячейку БД. Любой параметр, любая информация, которая была записана в журнальный файл, обязательно будут занесены в БД. Кроме того, каждому сообщению, которое копируется из журнального файла в базу данных, присваивается идентификационный номер. Это даёт возможность осуществлять быстрый поиск сообщений в базе данных.

Поскольку СУБД PostgreSQL, которая используется для создания БД «Динамика», поддерживает различные API [9], в качестве языка для написания программы qserv-stat.py был выбран высокоуровневый язык программирования Python. Основной задачей явилось создание программы для разбора журнального файла, которая бы имела понятный и легко читаемый код, что дало бы возможность в будущем при внесении изменений в параметры журнального файла также быстро внести изменения в файл qserv-stat.py. Именно поэтому был выбран язык программирования высокого уровня. Для того чтобы программа, написанная на языке Python, могла подсоединяться к СУБД PostgreSQL, использовался специально предназначенный для этого модуль PyGreSQL-3.8.

Программа qserv-stat.py для разбора журнального файла написана исходя из описания различных типов сообщений, которое содержится в документе «Руководство системного программиста (администратора) системы управления прохождением задач МВС-1000/М (версия 2.01)». Исходя из этого описания, разработана структура таблицы «messages», которая является основной в БД «Динамика». Установлено, что в журнальном файле содержится информация по 29 различным параметрам, соответственно столько же столбцов содержит таблица «messages». Кроме того, на основании данного описания была разработана структура таблицы «tasks», которая содержит всю информацию о каждой поступившей в систему задаче.

Алгоритм работы программы qserv-stat.py следующий. Сначала программа подсоединяется с помощью модуля PyGreSQL-3.8. к БД «Динамика», затем открывает журнальный файл и начинает разбирать каждое отдельное сообщение. На основании времени и даты составляется идентификационный номер. Далее определяется тип сообщения, и на основании

этого производится запись каждого параметра в базу данных. Заполнение таблиц «messages» и «tasks» происходит параллельно.

Таблица 1

Структура таблицы «messages»

Описание	Имя столбца	Тип записи	Варианты записи
номер сообщения (Primary Key)	id_mes	bigint	
дата и время	date_m	timestamp	
тип события	type_m	varchar (5)	*, +, -,!
тип сообщения	type_inf	varchar (15)	start New, UserSum, new_mode, RCV, SND, queue, run, term, fnsh, cont, stop, delete, error, sys_del, exit, locked, smth_else
пользователь	user_name	varchar (40)	
имя задачи	task	varchar (40)	
идентификатор процесса	pid	integer	
требуемое число процессоров для задачи	cpu	integer	
требуемое время для задачи	task_time	float	
квант времени для фоновой задачи	quant	float	
время счёта	work_time	float	
цена задачи	task_vol	float	
прием запроса с номером	TAG	integer	
параметры запроса	parameter	varchar	
код возврата	code	integer	
код завершения	type_stop	integer	0, 1, 2, 3, 4
число доступных процессоров	total	integer	
число свободных процессоров	free	integer	
шкала классов режима	scl	integer	
общее число планируемых процессоров	ttl	integer	
максимальное время отладочной задачи	emin	float	
действие, при котором произошла ошибка	action_err	varchar (20)	run, queue, delete
имя файла-паспорта задачи	name_fp	varchar (100)	
номер машины	machine_id	integer	
исходное сообщение	message	varchar (200)	

V. Таблица «messages»

Таблица «messages» является важнейшей таблицей в БД, так как она хранит в себе все сообщения, записанные в журнальный файл. Благодаря этой таблице, появляются широкие возможности для сбора статистики.

Таблица «messages» состоит из 29 столбцов. В табл. 1 указаны наиболее важные поля, которые содержат основную журнальную информацию. Разборка сообщения из журнального файла и дальнейшая запись параметров в таблицу

«messages» производит программа qserv-stat.py. Все параметры подробно описаны в инструкции для администраторов. Кроме информации, которая содержится в журнальном файле, каждой записи в таблице «messages» присваивается уникальный идентификационный номер. Также в столбце «message» хранится сообщение журнального файла целиком в первоначальном виде.

VI. Таблица «tasks»

Таблица «tasks» хранит в себе всю журнальную информацию по каждой задаче. При обработке журнального файла программа qserv-stat.py определяет все сообщения, которые несут в себе информацию о ходе выполнения задач. После того как сообщение разбирается на отдельные параметры, программа добавляет полученную информацию как в таблицу «messages», так и в таблицу «tasks». По сути, все сообщения, которые относятся к задачам, делятся на три вида: сообщение о постановке в очередь (queue), сообщение о начале выполнения (run), сообщение о завершении выполнения задачи (term, fnsh, cont, stop, delete). Программа создаёт новые строки в таблице «tasks» в случае сообщения типа queue и добавляет информацию в уже существующие строки в случае сообщения типа run, term, fnsh, cont, stop, delete. Структура таблицы «tasks» приведена в табл. 2.

Таблица 2

Структура таблицы «tasks»

Описание	Имя столбца	Тип записи	Варианты записи
имя пользователя	user_name	varchar (40)	
имя задачи	task	varchar (40)	
идентификатор процесса	pid	integer	
требуемое число процессоров для задачи	cpu	integer	
квант времени для фоновой задачи	quant	float	
требуемое время для задачи	task_time	float	
дата и время постановки в очередь	queue_time	timestamp	
дата и время запуска задачи	run_time	timestamp	
дата и время завершения задачи	term_time	timestamp	
код завершения	type_stop	integer	0, 1, 2, 3, 4, delete
номер машины	machine_id	integer	

Рассмотрим подробнее наиболее важные из столбцов, которые представлены в таблице.

1. В столбце «user_name» хранится имя пользователя, от чьего имени была запущена данная задача.

2. В столбце «task» хранится имя задачи.

3. В столбце «pid» хранится номер идентификатора задачи, который присваивается ей в момент начала её выполнения и при появлении в журнальном файле сообщения типа run.

4. В столбце «cpu» хранится число запрошенных задачей процессоров.

5. В столбце «quant» хранится значение кванта времени для фоновой задачи.

6. В столбце «type_stop» хранится код завершения задачи.

VII. Установка подсистемы «Динамика»

Для того чтобы установить подсистему «Динамика», прежде всего необходимо инсталлировать на компьютер СУБД PostgreSQL (версии 8.2 и выше). Затем нужно в СУБД PostgreSQL установить саму БД «Динамика». По сути, требуется установка только структуры таблиц «messages» и «tasks», основная информация из журнального файла в таблицы будет вводиться потом. В файле stat.backup хранятся структуры таблиц «messages» и «tasks» (названия полей и т. д.), а также процедуры на языке PL/pgSQL, которые позволяют собирать и обрабатывать статистическую информацию. БД

«Динамика» можно установить из файла stat.backup, используя как стандартный терминал для работы с СУБД, так и программу «pgAdmin» (которая доступна вместе с СУБД). В программе «pgAdmin III» устанавливать нужно следующим образом: выбрать в выпадающем окне «Tools» функцию «Restore», далее в открывшемся окне указать путь к файлу stat.backup и нажать «OK». После этого БД «Динамика» будет установлена. Её рабочее название в СУБД PostgreSQL будет БД «stat».

Затем следует установить на компьютер интерпретатор языка Python (например, Python 2.5, который можно скачать с официального сайта www.python.org), так как скрипты для загрузки информации из журнальных файлов в БД, а также для обработки статистики в БД написаны именно на этом языке. Затем нужно будет установить модуль PyGreSQL, который необходим для соединения Python с СУБД PostgreSQL (скачать модуль PyGreSQL можно бесплатно с официального сайта www.pygresql.org).

После этого необходимо заполнить информацией из журнальных файлов qserv.log таблицы в БД «Динамика». Для этого нужно воспользоваться программой qserv-stat.py. Для того чтобы qserv-stat.py записала всю информацию из файла qserv.log в БД, требуется, чтобы этот файл находился в той же директории, что и сама программа, а также нужно указать имя журнального файла в самой программе. Последняя строчка кода программы содержит имя считываемого файла, например:

Qserv-Stat («qserv2128.log», 1)

Здесь qserv2128.log — это имя считываемого файла. Кроме того, необходимо прописать путь к БД, по умолчанию он указан в программе как:

con = pg.connect ('stat', 'localhost', 5432, None, None, 'postgres', 'post1')

Если будет изменено имя БД, её владелец и т. д., параметры, указанные в программе, необходимо будет привести в соответствие с параметрами в базе данных. После успешного запуска qserv-stat.py таблицы в БД «Динамика» будут заполнены информацией из журнального файла.

Программы на языке Python для статистической обработки информации в БД (python-task_1.py, python-task_2.py, python-task_3.py, python-task_4.py, python-task_5.py) удобно запускать, используя стандартную для Python 2.5 утилиту «IDLE (Python GUI)». Точно так же следует запускать программу qserv-stat.py.

VIII. Программы для статистического анализа информации

Как было сказано выше, основной задачей подсистемы «Динамика» является мониторинг производительности кластеров суперкомпьютерного центра. Возможность постоянного наблюдения за параметрами системы позволяет администратору и инженеру вовремя обнаружить любое существенное отклонение от оптимальных показателей и принять меры для исправления возникших неполадок. Также очень важно записывать в базу данных уже собранную информацию, для того чтобы иметь возможность собирать статистику по различным параметрам за длительный период времени и сравнивать эффективность работы системы при разных конфигурациях, настройках [10]. Именно в этих целях были написаны программы для статистического анализа информации, которая хранится в базе данных «Динамика».

Для сбора статистической информации и мониторинга параметров высокопроизводительных вычислительных систем необходимо использовать методы, позволяющие свести вмешательство в работу суперкомпьютеров к минимуму. Именно поэтому использование сервера очередей qserver в качестве источника информации для журнального файла является оптимальным. Параметры, которые записываются этим процессом, являются тем минимумом, без которого невозможно организовать эффективную работу кластеров. С другой стороны, никаких дополнительных параметров, при запросе которых могла бы снизиться производительность системы, здесь не используется. Задача подсистемы «Динамика» заключается именно в создании дополнительных программ для оценки эффективности работы суперкомпьютеров без снижения общей производительности.

На данный момент для статистического анализа информации, которая хранится в подсистеме «Динамика», написаны следующие 5 программ на языке Python.

1. python-task_1.py — задача программы — определить среднее время ожидания задачи в очереди на запуск в зависимости от числа запрошенных процессоров.

2. python-task_2.py — задача программы — определить среднее время ожидания задачи в очереди на запуск в зависимости от типа задачи (существует два типа задач: пакетные и фоновые).

3. python-task_3.py — задача программы — собрать статистику по прерванным задачам в зависимости от времени обработки, которое было ими указано.

4. python-task_4.py — задача программы — собрать статистику по количеству задействованных процессоров в системе и определить среднее число задействованных процессоров за сутки.

5. python-task_5.py — задача программы — собрать статистику по коду завершения задач. Данный код указывается для каждой задачи в сообщении о завершении задачи (сообщение типа stop).

IX. Структура работы программ

Все созданные программы для статистического анализа информации имеют сходную структуру. Как уже было сказано, программы состоят из двух модулей, один из которых написан на языке высокого уровня Python, а другой — на процедурном языке PL/pgSQL, являющемся аналогом процедурного языка PL/SQL для СУБД PostgreSQL. Использование PL/pgSQL позволяет наиболее оптимальным образом собрать интересующую информацию в базе данных «Динамика». Другая же часть программы, написанная на языке Python, позволяет дообработать собранную PL/pgSQL информацию и вывести её пользователю в наиболее удобном виде, так как язык Python обладает большим набором возможностей, чем специализированный процедурный язык PL/pgSQL [11].

Каждая программа для статистического анализа информации базы данных «Динамика» выполняет один и тот же набор действий.

1. Необходимо запустить программу python-task_№. py (это будет общее название множества программ python-task_1.py, python-task_2.py, python-task_3.py и т. д.).

2. После запуска python-task_№. py программа подключается к базе данных «Динамика», используя специальный модуль PyGreSQL-3.8, который должен быть установлен до начала работы программы.

3. После подключения к БД программа python-task_№. py запускает подпрограмму task_№, которая написана на процедурном языке PL/pgSQL (номер запущенной подпрограммы соответствует номеру программы, например, программа python-task_3.py запускает подпрограмму task_3.py).

4. Подпрограмма task_№ собирает статистическую информацию из таблиц базы данных «Динамика», используя в своём коде цикл вида:

FOR row_data IN SELECT * FROM имя_таблицы LOOP

...

END LOOP.

1. Данный цикл даёт возможность собрать всю необходимую информацию, просмотрев интересующую таблицу всего один раз. Таким образом, количество действий при сборе статистической информации подпрограммой task_№ зависит как $O(n)$, где n — число строк в таблице.

2. После окончания работы подпрограмма task_№ передаёт собранную информацию программе python-task_№. py, которая дообрабатывает полученную информацию, а затем выводит результат пользователю.

В качестве источника данных был использован журнальный файл qserv.log, который хранил в себе информацию о работе суперкомпьютера МВС-15000ВМ за полтора месяца. В данном файле хранится около 120 000 сообщений о работе порядка 10 000 задач. Подобное количество журнальной информации вполне достаточно для проведения статистического анализа.

Унификация структуры работы программ для статистического анализа информации даёт возможность в будущем при изменении структуры журнального

файла быстро модифицировать код данных программ.

X. Среднее время ожидания задачи в зависимости от числа запрошенных процессоров

Рассмотрим программу для вычисления среднего времени ожидания задачи в зависимости от числа запрошенных процессоров. Данная программа называется python-task_1.py в подсистеме «Динамика». Программа собирает и обрабатывает статистическую информацию из базы данных с целью выяснить, как именно зависит продолжительность ожидания задачи (с момента её постановки в очередь и до начала её выполнения) от числа запрошенных процессоров, которые программа запрашивает в момент постановки в очередь.

Таблица 3

Среднее время ожидания задачи в зависимости от числа запрошенных процессоров

Интервал, кол-во процессоров	Среднее время ожидания, мин	% задач в интервале от общего числа задач
1	25	0,8
2–3	32	35,4
4–7	23	18,3
8–15	157	15,7
16–31	175	10,4
32–63	249	7,1
64–127	454	6,8
128–255	809	4,0
256–511	707	1,2
512–922	348	0,4

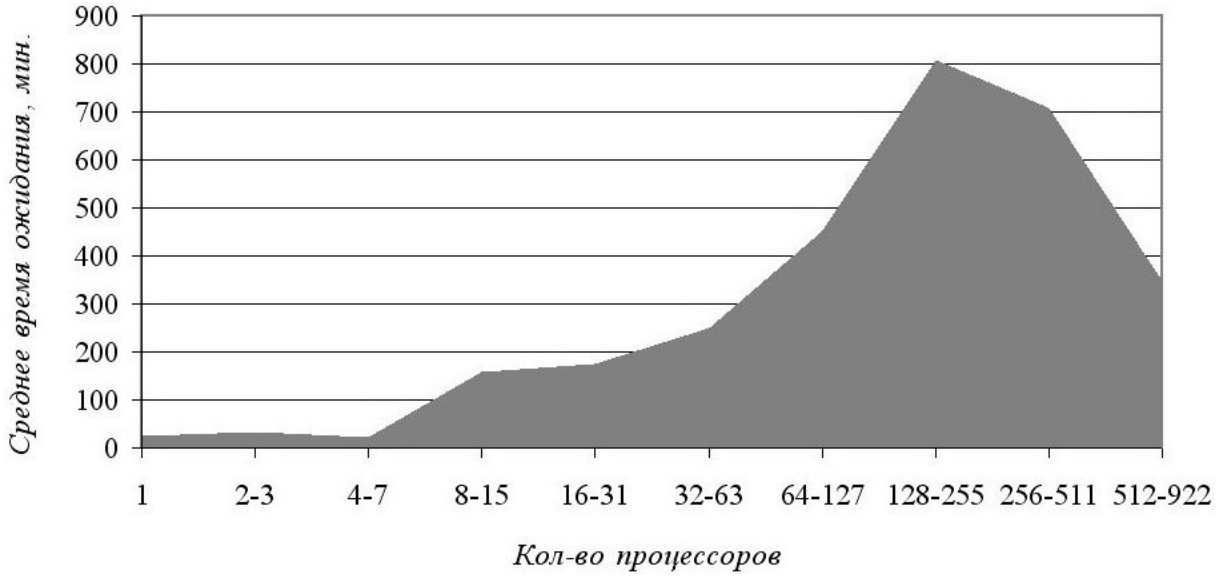

Рис. 1. Среднее время ожидания задачи в зависимости от числа запрошенных процессоров

В результате работы программы были получены данные, приведённые в табл. 3. Количество процессоров, запрошенных различными программами, поделено на интервалы, которые приведены в первом столбце. В следующем столбце указано среднее время ожидания для выбранного интервала, проходящее перед тем, как задача будет запущена. В последнем столбце указан процент задач в интервале от общего числа обработанных задач.

На рис. 1 хорошо видно, что сначала при небольшом количестве запрошенных процессоров (от 1 до 7) среднее время ожидания колеблется в районе 20–30 минут. С увеличением числа запрошенных про-

цессоров среднее время ожидания возрастает, но до определённого предела (в районе 256–511 процессоров). Затем следует спад, связанный с работой администраторов системы, которые способствуют началу решения крупных задач.

На рис. 2 показано отношение (в процентах) задач в интервале от общего числа задач в зависимости от числа запрошенных процессоров. На данном графике хорошо видно, что число задач, для которых был запрошен всего один процессор, очень мало — всего 0, 8% от общего числа задач. По сути, такой малый процент обусловлен тем, что для любой задачи, отправляемой на выполнение, система выделяет как минимум один вычислительный модуль. В суперкомпьютерах вычислительные модули содержат как минимум (в зависимости от модели) два процессора. Поэтому запрос одного процессора для выполнения задачи можно считать не совсем корректным. Именно по этой причине так мало задач, для которых был запрошен всего один процессор. Однако даже этот небольшой процент говорит о том, что не все пользователи суперкомпьютерного центра, отправляющие свои задачи на выполнение, имеют необходимое представление о структуре суперкомпьютеров и о правилах установки параметров для выполнения задач. Далее на графике следует интервал от 2 до 3 запрошенных процессоров, для которого доля обработанных задач является максимальной — 35, 4%. То есть наибольшее число задач запрашивает минимально возможное число выделяемых процессоров — два процессора. Поскольку при запросе одного процессора всё равно выделяется два, то можно к 35, 4% прибавить 0, 8%, и тогда получится, что 36, 4% задач от общего числа запрашивают от 1 до 3 процессоров. С увеличением числа запрошенных процессоров число задач монотонно уменьшается. Доля самых крупных задач, которым необходимо для работы наибольшее число процессоров (интервал от 512 до 922), составляет всего 0, 4%. В среднем такие задачи ожидают 348 минут.

Рис. 2. Отношение (в %) количества задач в интервале от общего числа задач в зависимости от числа запрошенных процессоров

XI. Среднее время ожидания в зависимости от типа задачи

Рассмотрим программу для вычисления среднего времени ожидания задачи в зависимости от типа задачи. Данная программа носит имя python-task_2.py в подсистеме «Динамика». В журнальном файле задачи делятся на два типа: пакетные и фоновые. Пакетные задачи после постановки в очередь запускаются только один раз и считаются в системе без перерывов до полного завершения. В случае постановки фоновой задачи в очередь на счёт, помимо информации о необходимом количестве процессорного времени, также указывается квант времени, который определяет минимальное время счёта для данной задачи. По истечении кванта времени задача может быть либо остановлена и вновь поставлена в очередь, либо счёт может быть продолжен. Общее время счёта для фоновой

задачи по её завершению должно быть равно указанному процессорному времени.

Алгоритм работы программы построен по общему принципу устройства всех программ по сбору статистической информации. Сначала запускается программа python-task_2.py, которая подсоединяется к базе данных «Динамика» и запускает функцию task_2 на языке PL/pgSQL. Далее функция начинает просматривать всю таблицу «tasks», используя для этого следующий цикл:

FOR row_data IN SELECT * FROM tasks LOOP

<тело цикла>

END LOOP;

В теле цикла для начала происходит определение, к какому типу (пакетная или фоновая) относится выбранная из таблицы задача. Для этого используется параметр «quant» таблицы «tasks». В случае, если это фоновая задача, в параметре «quant» будет указано числовое значение, отличное от нуля. Если же это пакетная задача, то «quant» будет иметь значение, равное нулю.

После определения того, к какому типу относится выбранная задача, производится её проверка. Поскольку программа собирает статистику по времени ожидания в зависимости от типа, то каждая выбранная задача должна иметь записи о времени постановки в очередь и о времени запуска. Только при выполнении этих двух условий может быть рассчитано время ожидания. Время постановки в очередь указывается в столбце «queue_time». Время запуска задачи указывается в столбце «run_time». Соответственно происходит проверка, чтобы в обоих этих столбцах значения были не равны NULL.

После определения типа задачи и её проверки к счётчику данного типа задач прибавляется единица. Затем рассчитывается время ожидания путём вычитания времени постановки в очередь из времени запуска задачи. После этого время ожидания прибавляется к общему времени ожидания для данного типа задач. По окончании цикла мы получаем информацию о количестве задач для каждого типа и об общем времени ожидания для каждого типа задачи.

В конце своей работы функция task_2 передаёт собранную информацию обратно python-task_2.py. Данная программа делит общее время ожидания на количество задач для каждого типа и, таким образом, вычисляет среднее время ожидания для пакетных и фоновых задач.

После запуска программы были получены следующие результаты (рис. 3):

— среднее время ожидания для пакетной задачи равно 150 мин;

— среднее время ожидания для фоновой задачи равно 136 мин.

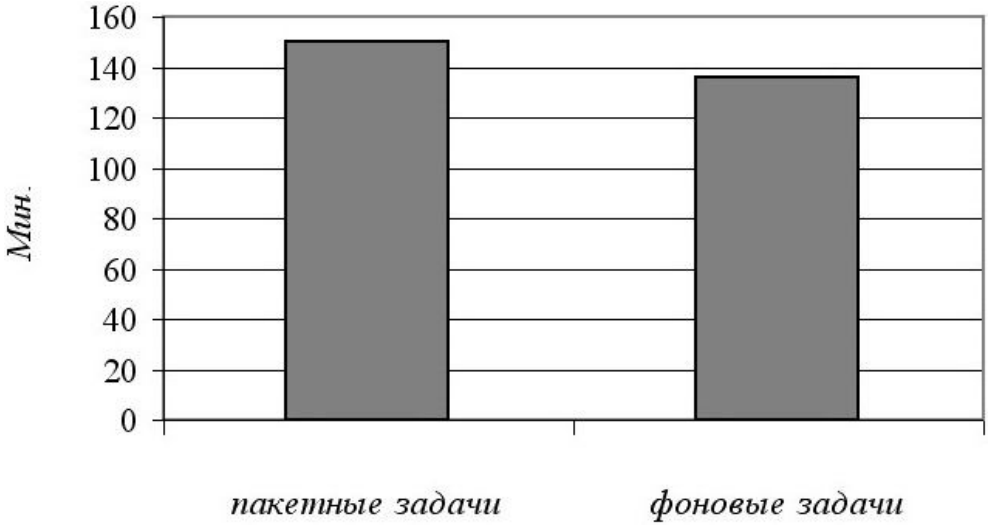

Рис. 3. Среднее время ожидания в зависимости от типа задачи

Также программа собрала статистическую информацию о численном соотношении между пакетными и фоновыми задачами (рис. 4):

— доля пакетных задач от общего числа задач равно 97,2%;

— доля фоновых задач от общего числа задач равно 2,8%.

Рис. 4. Доли пакетных и фоновых задач

На диаграмме хорошо видно, что доля фоновых задач очень мала. Это можно объяснить тем, что далеко не каждую пользовательскую задачу целесообразно запускать в фоновом режиме. Во-первых, задача может быть не приспособлена к обработке данных в несколько этапов, так как часть данных может быть потеряна при остановке и повторном возобновлении работы. А во-вторых, возможно, ощутимая часть времени будет уходить на загрузку и выгрузку данных из памяти суперкомпьютера. Поэтому, конечно, пакетный режим работы является более предпочтительным для задач, требующих серьёзных компьютерных ресурсов. Именно для таких задач чаще всего используются высокопроизводительные вычислительные системы суперкомпьютерного центра.

XII. Статистика по прерванным задачам в зависимости от времени обработки

Рассмотрим программу для сбора статистики по прерванным задачам в зависимости от запрошенного времени её обработки. Данная программа носит имя python-task_3.py в подсистеме «Динамика». Статистика по прерванным задачам является важной информацией, так как она показывает, какое количество процессорного времени фактически было потрачено впустую. Завершившиеся и прерванные задачи можно различать по коду завершения. Код завершения по каждой обработанной задаче указывается в сообщении журнального файла типа «stop». Коды завершения бывают семи различных типов, но в данном случае программой используются следующие два типа:

— код завершения «1» указывает на то, что задача была досчитана до конца и завершилась сама, такая задача называется завершившейся;

— код завершения «4» указывает на то, что задача полностью исчерпала своё время, которое было указано при постановке задачи в очередь, и была принудительно завершена, такая задача называется прерванной.

Таблица 4

Статистика по прерванным задачам в зависимости от времени обработки

Интервал, мин	Завершив. задачи, проц. *мин	Прерванные задачи, проц. *мин	% интервал времени от всего времени	% в интер. по прерв. задачам	% прерв. в инт. задач от всего времени
0–5	42360	11194	0,09	20,90	0,019
5–10	68880	11904	0,13	14,74	0,020
10–20	209478	109904	0,53	34,41	0,184
20–40	622302	142570	1,28	18,64	0,238
40–80	481260	216832	1,17	31,06	0,362
80–160	1525808	582580	3,52	27,63	0,974
160–320	3564012	1391620	8,28	28,08	2,326
320–640	21830864	8438450	50,58	27,88	14,101
640–1280	7246740	3629416	18,18	33,37	6,065
1280–2560	5747354	3967324	16,23	40,84	6,630

В результате работы программы были получены результаты, которые приведены в табл. 4. На рис. 5 представлено отношение (в процентах) процессорного времени в каждом интервале ко всему процессорному времени в зависимости от интервала. График строится на основании данных, приведённых в таблице. По оси X откладывается значение из столбца «Интервал, мин», по оси Y откладывается соответствующее значение из столбца «% интервала времени от всего времени». В столбце «% интервала времени от всего времени» указывается доля в процентах процессорного времени (сумма процессорного времени по завершившимся и прерванным задачам) в интервале к общему процессорному времени.

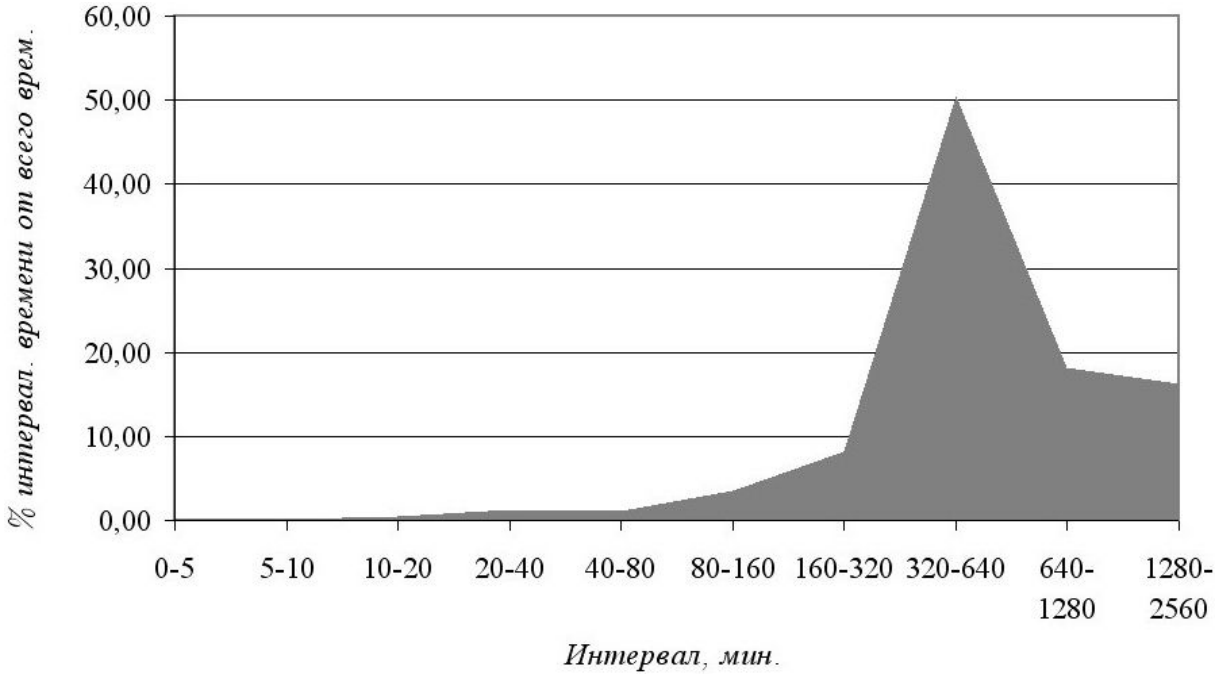

Рис. 5. Отношение (в %) процессорного времени в каждом интервале ко всему процессорному времени в зависимости от интервала

На рис. 6 представлено отношение (в процентах) процессорного времени по прерванным задачам к общему процессорному времени в интервале в зависимости от интервала.

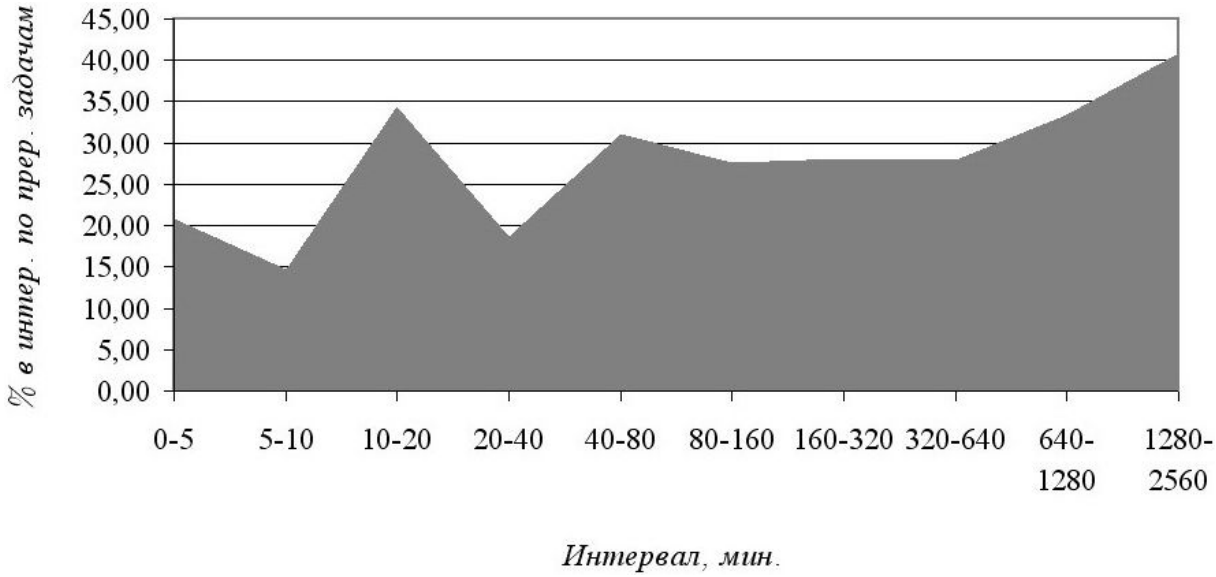

Рис. 6. Отношение (в %) процессорного времени по прерванным задачам к общему процессорному времени в интервале в зависимости от интервала

На рис. 7 представлено отношение (в процентах) процессорного времени по прерванным задачам в каждом интервале ко всему процессорному времени в зависимости от интервала.

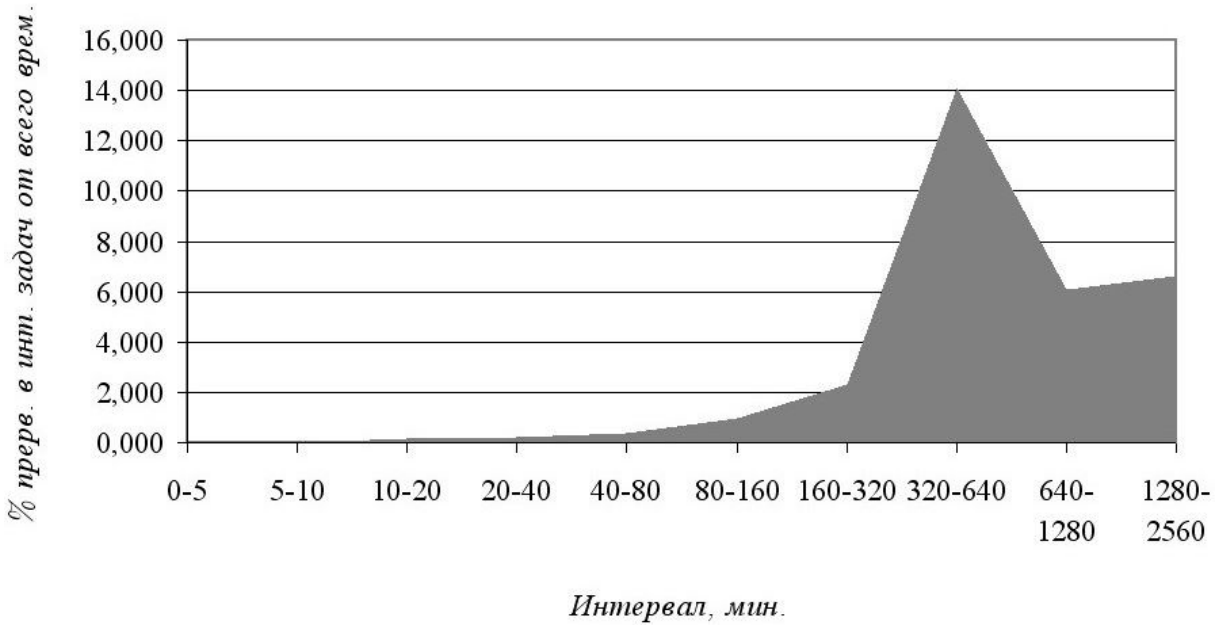

Рис. 7. Отношение (в %) процессорного времени по прерванным задачам в каждом интервале ко всему процессорному времени в зависимости от интервала

Также программа рассчитала, что доля процессорного времени по всем прерванным задачам по отношению ко всему процессорному времени составляет 30,9%.

XIII. Статистика по количеству задействованных процессоров

Рассмотрим программу для сбора статистики по количеству задействованных процессоров в зависимости от даты, когда выбранный суперкомпьютер был задействован. Данная программа носит имя python-task_4.py в подсистеме «Динамика». Статистика по среднему количеству задействованных процессоров за продолжительный период времени (например, за сутки) является чрезвычайно важным показателем работы системы, так как информирует администратора, насколько эффективно используются ресурсы суперкомпьютерного центра. Решение о запуске задач принимает сервер очередей параллельных задач qserver, который функционирует на управляющей ЭВМ. Именно этот процесс отвечает за то, насколько эффективно используется процессорное время. Напомним, что qserver также ведёт журнальный файл qserv.log, из которого подсистема «Динамика» получает информацию о событиях, происходящих в системе.

Алгоритм работы программы построен на анализе сообщений из таблицы «messages», содержащих информацию о количестве задействованных процессоров. Основную часть сбора и обработки информации в программе python-task_4.py выполняет функция task_4, написанная на процедурном языке PL/pgSQL. Программа строит массив, в котором записывает, сколько процессоров было задействовано и на протяжении какого времени. Этот массив строится с использованием даты и времени, указываемых в каждой строке таблицы, а также информации о числе задействованных процессоров на момент создания сообщения.

Перед началом сбора статистики по количеству задействованных процессоров функция сортирует записи в таблице «messages» точно в том порядке, в каком они были записаны в журнальном файле qserv.log. Рассмотрим основные журнальные сообщения, использующиеся при работе программы.

1. Сообщения типа «start», для которых в таблице «messages» в столбце «free» указывается число свободных процессоров на момент старта, а в столбце «total» ука-

зывается общее число доступных процессоров. Поскольку программа изначально не знает, чему равно общее число доступных процессоров, она в начале своей работы находит сообщение типа «start», откуда и берёт дату и время записи, общее число доступных процессоров, общее число свободных процессоров.

2. Сообщения типа «run», для которых в таблице «messages» в столбце «cpu» указывается число выделенных процессоров для работы задачи. Программа использует дату и время записи сообщения, а также число выделенных процессоров, увеличивая на соответствующую величину число задействованных процессоров.

3. Сообщения типа «term», «fnsh», «cont», для них в таблице «messages» в столбце «cpu» указывается число высвобожденных процессоров из-за прекращения работы задачи. Программа использует дату и время записи сообщения, а также число освободившихся процессоров, уменьшая на соответствующую величину число задействованных процессоров.

В результате работы программы были получены результаты, изображённые на рис. 8. Данные отражают статистику по количеству задействованных процессоров более чем за месяц работы суперкомпьютера.

Средний процент задействованных процессоров от всех вычислительных мощностей за всё время составил 88%. Спады, наблюдаемые на графике, связаны с профилактическими работами.

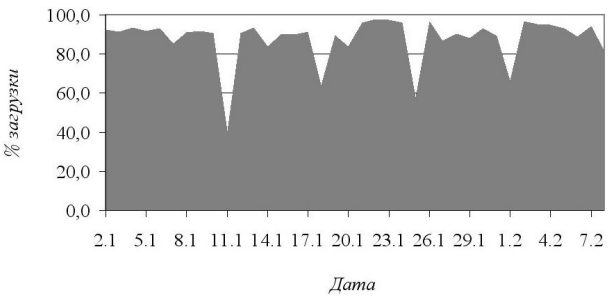

Рис. 8. Статистика по количеству задействованных процессоров

XIV. Статистика по коду завершения задач

Рассмотрим программу для сбора статистики по коду завершения задач. Данная программа носит имя python-task_5.py в подсистеме «Динамика». Программа собирает и обрабатывает статистическую информацию из базы данных с целью выяснить для каждого кода завершения, какой процент составляют задачи с данным кодом завершения от общего числа задач.

Код завершения по каждой завершившейся задаче указывается в сообщении журнального файла типа «stop». Коды завершения имеют следующую расшифровку:

0 — исчерпан квант времени фоновой задачи;

1 — задача досчитала до конца и завершилась сама;

2 — задача снята пользователем;

3 — задача снята администратором;

4 — задача полностью исчерпала своё время;

5 — по завершении задачи происходит повторный запуск;

delete — удаление задачи из очереди.

В результате работы программы были получены следующие результаты, представленные в табл. 5. В столбце «Количество задач» указано число задач, завершённых с соответствующим кодом за выбранный период времени (40 дней). В столбце «% от всех задач» указано соотношение в процентах количества задач с данным кодом завершения к общему числу завершившихся задач. Как видно из рис. 9, наибольший процент задач (72,9%) был успешно досчитан до конца.

Таблица 5

Статистика по прерванным задачам в зависимости от времени обработки

Код завершения	Количество задач	% от всех задач
0	137	1,5
1	6494	72,9
2	342	3,8
3	28	0,3
4	1262	14,2
5	46	0,5
delete	601	6,7

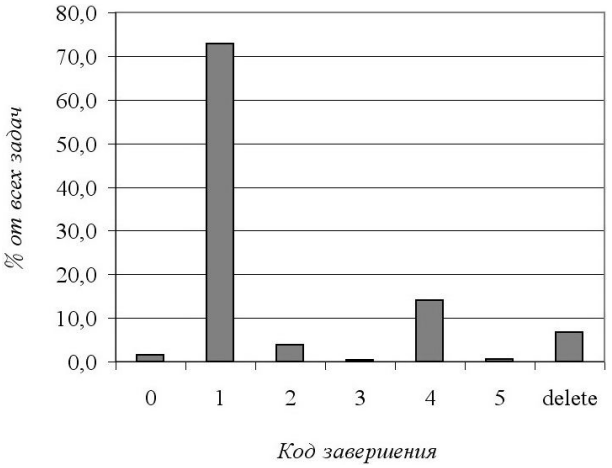

Рис. 9. Статистика по коду завершения задач

XV. Выводы

Для хранения и статистического анализа информации о потреблении вычислительных ресурсов программы «Университетский кластер» была создана подсистема «Динамика», состоящая из базы данных «Динамика» и программ для сбора и обработки информации. Данная БД построена на базе СУБД PostgreSQL 8.2, которая обеспечивает высокую производительность и предоставляет широкие возможности для создания различных функций и процедур. Программы для статистического анализа, написанные на языках Python и PL/pgSQL, используют алгоритмы, позволяющие обрабатывать большие массивы информации.

Продуманная структура базы данных и эффективная работа программ-приложений позволяют подсистеме «Динамика» решать следующие поставленные перед ней задачи.

1. Хранение информации о ходе работы различных суперкомпьютеров, входящих в состав МСЦ.

2. Хранение информации о потреблении вычислительных ресурсов различными задачами и пользователями.

3. Осуществление быстрого поиска любого события и любой информации, отражённых в журнальных файлах различных кластеров.

4. Статистический анализ информации о потреблении вычислительных ресурсов с целью контроля эффективности работы системы.

Подсистема «Динамика» даёт возможность анализировать эффективность работы компонентов системы за различные периоды времени. Мониторинг параметров производительности суперкомпьютеров МСЦ помогает выявить различные системные сбои и ошибки на самых ранних этапах и позволяет избежать снижения производительности.

Литература

1. *Уорсли Д., Дрейк Д.* PostgreSQL. Для профессионалов. — СПб.: Питер, 2003.

2. *Стоунз Р., Нейл М.* PostgreSQL. Основы. — М.: Символ-Плюс, 2002.

3. *Аткинсон Л.* MySQL. Библиотека профессионала. — М.: Вильяме, 2002.

4. *Дюк Р.* База данных Oracle 10g: Администрирование. — М.: НОУ УКЦ ФОРС, 2005.

5. *Каучмэн Д., Швинн У.* Oracle8i Certified Professional DBA. Подготовка администраторов баз данных. — М.: Лори, 2002.

6. *Гешвинде Э., Шениг Г.* PostgreSQL. Руководство разработчика и администратора. — М.: ДиаСофтЮП, 2002.

7. *Васвани В.* Полный справочник по MySQL. — М.: Вильяме, 2006.

8. *Борзов А.С.* PostgreSQL: настройка производительности. — М.: Мир, 2002.

9. *Россум Г., Дрейк Д., Откидач Д.С.* Язык программирования Python. — М.: Вильяме, 2001.

10. *Касперски К.* Техника оптимизации программ. Эффективное использование памяти. — М.: BHV, 2003.

11. *Брандт З.* Анализ данных. Статистические и вычислительные методы для научных работников и инженеров. — М.: Мир, 2003.

Поступила в редакцию 02.02.2009.