CAEBeans Broker: брокер ресурсов системы CAEBeans*

А.В. Шамакина

В работе дано описание CAEBeans Broker – автоматизированной системы регистрации, анализа и предоставления ресурсов распределенной вычислительной среды. Приводятся логическая структура, архитектура очередей и алгоритм планирования CAEBeans Broker. Описаны аспекты реализации грид-сервиса CAEBeans Broker в контейнере WSRFLite.

1. Введение

В настоящее время важную роль при разработке новых технологических линий и производственных технологий играют компьютерное моделирование и виртуальные испытательные стенды, зачастую требующее значительного количества вычислительных ресурсов и времени. В качестве примера можно привести классическую задачу обтекания профиля крыла самолета в аэродинамике или использование в промышленности стендов для задачи овализации труб при их закалке и последующем отпуске [1]. Такой подход позволяет существенно повысить точность анализа проектных вариантов продукции, значительно сократить стоимость опытно-конструкторских работ и проводить виртуальные эксперименты, которые в реальности выполнить невозможно.

Для решения данной проблемы предлагается использовать программные системы класса CAE (Computer Aided Engineering), позволяющие эффективно осуществлять суперкомпьютерное моделирование сложных технологических процессов, а также вычислительные ресурсы грид.

В качестве перспективного подхода к решению задач с помощью инженерных пакетов в распределенных вычислительных средах предлагается использовать технологию CAEBeans [2, 3]. В ее основе лежит обеспечение сервисно-ориентированного предоставления программных ресурсов базовых компонентов CAE-систем и формирование деревьев проблемно-ориентированных оболочек, инкапсулирующих процедуру постановки и решения конкретного класса задач. Технология CAEBeans автоматизирует: декомпозицию задач на типовые подзадачи; поиск вычислительных ресурсов; постановку задач соответствующим базовым компонентам CAE-систем; мониторинг хода решения задач; предоставление результатов решения задачи пользователю.

Система CAEBeans представляет собой совокупность программных средств, данных и аппаратных ресурсов, ориентированных на поддержку технологии CAEBeans. CAEBeans Broker – один из компонентов, входящих в систему CAEBeans. Брокер ресурсов представляет собой автоматизированную систему регистрации, анализа и предоставления ресурсов распределенной вычислительной среды.

Грид объединяет разнородные вычислительные ресурсы, использует отличные друг от друга стратегии доступа к ним. Научные, инженерные и коммерческие задачи, решающиеся посредством грид, предъявляют к системе различные требования. Вычислительные ресурсы обычно принадлежат нескольким организациям, имеющим свои правила управления ресурсами, использования и определения стоимости их для пользователей. Доступность и загруженность ресурсов также могут динамически изменяться во времени.

В соответсвии с этим актуальной проблемой является разработка систем управления ресурсами грид, нацеленных на эффективное решение задач и рациональное использование ресурсов в соответствии с выбранным алгоритмом планирования ресурсов в распределенных вычислительных средах.

В работе [4] рассматриваются пакетные и планирующие системы. К наиболее известным системам относится Condor [5]. Condor – это пакет программ для выполнения пакетных заданий

* Работа выполняется при финансовой поддержке программы СКИФ-ГРИД (договор № 2009-СГ-03) и Федеральной целевой программы (контракт № П2036).

на платформах UNIX. Главными особенностями Condor являются автоматическое нахождение ресурсов, распределение работ, установка контрольных точек и миграция процессов.

В переносимой системе PBS (Portable Batch System) [6] реализована пакетная организация очереди заданий и управления рабочей нагрузкой, первоначально разработанная для NASA. Планировщик заданий PBS позволяет локальным узлам устанавливать их собственную дисциплину планирования во времени и пространстве. PBS адаптирован к различным вариантам политики администрирования и обеспечивает расширяемую аутентификацию и модель защиты, регистрация заданий, отслеживание их выполнения и другие административные функций осуществляются посредством графического интерфейса.

Платформа Sun Grid Engine (SGE) [7] основана на программном обеспечении, разработанном фирмой Genias, известном как Codine/GRM. В SGE задания находятся в зоне ожидания, а очереди на серверах обеспечивают сервисы для заданий. Потребитель вводит задание в SGE и объявляет профиль необходимых требований для его выполнения. SGE определяет заданию соответствующую очередь и распределяет его либо с высшим приоритетом, либо с самым длинным временем ожидания, пробуя запускать новые задания на наиболее соответствующей или наименее загруженной очереди.

Средство распределения нагрузки LSF (Load Sharing Facility) [8] – это коммерческая система от Platform Computing Corp. В настоящее время является наиболее широко используемой коммерческой системой для управления выполнением заданий. LSF включает распределенный механизм разделения нагрузки и программное обеспечение для организации очередей заданий, которое управляет, контролирует и анализирует ресурсы и рабочие нагрузки на сети разнородных компьютеров, имея при этом возможности обеспечения отказоустойчивости.

Многие системы управления ресурсами грид (например, Legion [9], Condor [5], AppLeS PST [10], NetSolve [11], PUNCH [12], XtremWeb [13] и т.д.) используют простые схемы распределения, когда компонент, отвечающий за распределение, решает, какие задачи должны быть выполнены на каком ресурсе, используя функции стоимости, задаваемые системными параметрами. Целью таких систем распределения является увеличение пропускной способности системы, ее загруженности и уменьшения времени выполнения задач, а не увеличение рентабельности ресурсов и приложений.

В работе [14] предложено использование экономического подхода к планированию и распределению ресурсов, когда решения о распределении ресурсов производятся динамически и зависят от текущих требований пользователей. Подход реализован в системе GRACE [15]). Это – рыночная модель распределения ресурсов, когда цена каждого ресурса определяется потребностями в нем пользователей и его доступностью. Таким образом, в системе грид пользователь конкурирует с другими пользователями и владелец ресурса с другими владельцами ресурсов. Экономический подход позволяет успешно управлять децентрализованными и гетерогенными ресурсами так, как это происходит в реальной экономике. Экономические системы управления ресурсами грид динамически определяют наилучшие ресурсы, учитывая их цену и производительность, и распределяют задачи на этих ресурсах так, чтобы удовлетворить потребности пользователей.

Однако использование в системе CAEBeans вышеперечисленных брокеров ресурсов и планировщиков является невозможным, поскольку они работают на более низком системном уровне, чем компонент CAEBeans Broker. Кроме того, ни одна из ранее созданных систем управления ресурсами не удовлетворяет специфике системы CAEBeans [16], в рамках которой необходимо учитывать не только характеристики ресурсов, но и наличие установленных инженерных пакетов, количество доступных лицензий на них и др.

В настоящей статье рассмотрены архитектура и аспекты реализации компонента CAEBeans Broker. Статья организована следующим образом. В разделе 2 приводится описание логической структуры, архитектуры очередей и алгоритма планирования CAEBeans Broker. В разделе 3 обсуждается реализация грид-сервиса CAEBeans Broker в контейнере WSRFLite. В заключении суммируются основные результаты, полученные в данной статье.

2. CAEBeans Broker

CAEBeans Broker – это компонент системы CAEBeans, который принимает задания от пользователя, согласовывает требования к ресурсам и направляет задания на подходящий вычислительный элемент. Можно выделить следующие основные задачи брокера ресурсов: обработка каталога ресурсов грид-среды; анализ запросов на предоставление ресурсов, поступающих от компонентных CAEBean; сбор и предоставление информации об актуальном состоянии грид-среды.

Реализация CAEBeans Broker производится на языке программирования Java в виде грид-сервиса на основе Web Services Resource Framework (WSRF), спецификаций дополняющих стандарты web-служб и позволяющих устанавливать связь между Web-службами и ресурсами (WS-Resource, WS-ResourceProperties, WS-ResourceLifetime, WS-BaseFaults, WS-ServiceGroup, WS-Notification). Данный подход обеспечивает независимость компонента от вычислительной платформы, предоставление полной информации о текущем состоянии экземпляра сервиса, а также поддерживает возможность надежного и безопасного исполнения, управление временем жизни; рассылку уведомлений об изменении состояния экземпляра сервиса, управление политикой доступа к ресурсам, управление сертификатами доступа и виртуализации.

2.1 Логическая структура CAEBeans Broker

Общая структура брокера ресурсов CAEBeans Broker приведена на рис. 1. Программный компонент представляет собой грид-сервис, предоставляющий интерфейс управления ресурсами, содержащимися в каталоге ресурсов.

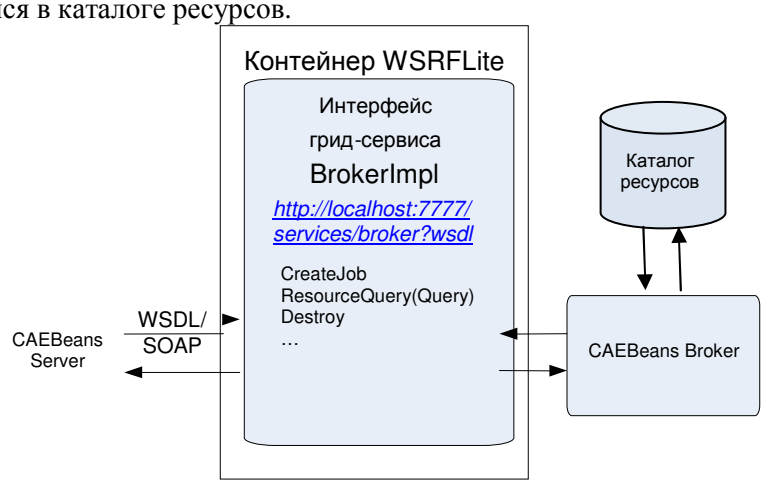

Рис. 1. Структура брокера ресурсов

В процессе работы брокера поиск требуемых ресурсов выполняется для нескольких заданий одновременно. При старте компонента CAEBeans Broker производится регистрация на определенном порте грид-сервиса, к которому при помощи протокола SOAP 1.2 обращается компонент CAEBeans Server для поиска подходящих ресурсов. Описание методов грид-сервиса CAEBeans Broker, а также протокола взаимодействия компонентов CAEBeans Server и CAEBeans Broker приводится в работах [17, 18].

2.2 Алгоритм планирования брокера ресурсов

Алгоритм планирования брокера ресурсов основывается на очередях к вычислительным ресурсам, архитектура которых описана в работе [18], и понятии физического и виртуального ресурсов.

При запуске грид-сервис CAEBeans Broker оперирует понятием физического ресурса. Под физическим ресурсом или целевой системой понимается совокупность аппаратных ресурсов вычислительной системы, операционной системы, инженерных пакетов и системных CAEBean, которые играют роль грид-оболочек над инженерными пакетами. На целевой системе в качест-

ве программного обеспечения промежуточного слоя установлен Unicore 6.0. Вся информация о целевых системах, лицензиях и заданиях хранится в каталоге ресурсов, который представляет собой базу данных.

Виртуальный ресурс – это часть физического ресурса, которая имеет собственную очередь заданий. В каждый момент времени на виртуальном ресурсе выполняется только одно задание. Это существенно облегчает процесс управления ресурсами.

Поступающие к CAEBeans Broker задания помещаются во входную очередь – *inputQueue*. Обработку заданий осуществляет одна из подсистем брокера ресурсов – планировщик. Планировщик ресурсов в цикле извлекает задания из очереди *inputQueue*, находит подходящий физический ресурс, формирует из него виртуальный ресурс, путем выделения физических узлов, количество которых определяется требованиями задания. Тем самым осуществляется первоначальное распределение физических ресурсов на виртуальные. Для каждого виртуального ресурса создается своя очередь заданий. Совокупность всех очередей к виртуальным ресурсам обозначим *virtualQueue*. Процесс создания виртуальных ресурсов продолжается до тех пор, пока не закончатся свободные физические ресурсы. Последующие задания из очереди *inputQueue* будут ставиться планировщиком в очереди к уже созданным виртуальным ресурсам: планировщик находит все виртуальные ресурсы, удовлетворяющие требованиям задания, ставит задание к каждому из них в очередь.

При обработке заданий планировщиком возможна ситуация, когда задание не может быть решено посредством имеющихся виртуальных ресурсов, но объединение вычислительных возможностей нескольких виртуальных ресурсов, находящихся на базе одного физического ресурса, позволит обеспечить решение данного задания. В данном случае планировщик определит задание в очередь *waitQueue*. На Рис. 2 приведены очереди к вычислительным ресурсам.

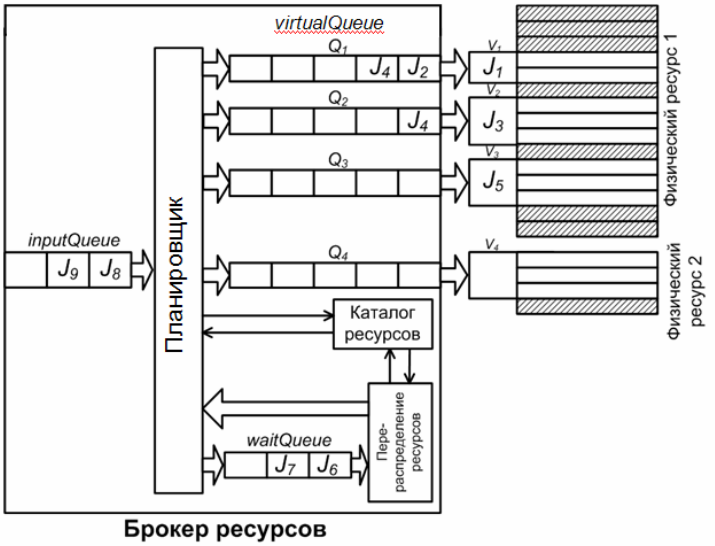

Рис. 2. Очереди к вычислительным ресурсам

Кроме того, планировщик осуществляет поиск подмножества виртуальных ресурсов, которые в объединении дают один большой ресурс для решения задания. Данная задача является обратной к известной «задаче о рюкзаке». Ведется поиск оптимального варианта: требуется отобрать некоторое подмножество виртуальных ресурсов с минимальным суммарным числом узлов, достаточным для решения задания. Искомое подмножество должно удовлетворять всем требованиям задания, включая программные. После нахождения такого подмножества производится маркировка всех виртуальных ресурсов, входящих в него. При дальнейшей обработке планировщиком заданий из очереди *inputQueue* распределение будет проводиться по всем ресурсам, кроме маркированных. При появлении следующего «большого» задания новой маркировки производиться не будет, чтобы предотвратить блокировку ресурсов. Планировщик ограничится помещением задания в очередь *waitQueue*.

Таким образом, после окончания вычислений на всех маркированных виртуальных ресурсах можно произвести объединение. Планировщик извлекает из очереди *waitQueue* задания и распределяет их заново по виртуальным ресурсам, включая задание, для которого выполнено

объединение. Если очередь *waitQueue* содержит задания, требующие нового объединения ресурсов, то производится новый процесс маркировки. При этом задание остается в очереди ожидания. Процесс перераспределения ресурсов с помощью механизма маркировки носит динамический характер.

Если всех физических ресурсов, имеющихся в наличие у CAEBeans Broker, не достаточно для выполнения некоторого задания, то компоненту CAEBeans Server отправляется соответствующая информация.

3. Реализация очередей брокера ресурсов

В многопроцессорных системах часто возникает проблема, связанная с возникновением блокировок. Проявляется она в ситуации, когда параллельные потоки конкурируют за доступ к некоторому синхронизированному ресурсу. Если доступ к ресурсу реализован с помощью обычной взаимоисключающей блокировки (mutex), то доступ к нему получает только один поток, а остальные ждут своей очереди. Если таких потоков несколько, то суммарное время ожидания всех потоков может существенно превысить время, которое требуется на непосредственное выполнение работы.

Решить проблему можно отказавшись от использования блокировок. Часто в роли общего ресурса является некоторая коллекция. Традиционный подход заключается в том, что коллекция оборачивается в синхронизирующую обертку, например, с помощью метода Collections.synchronizedCollection(). Это означает, что любой доступ к этой коллекции автоматически устанавливает блокировку, а следовательно, порождает возникновение блокировки.

В Java 5 был существенно расширен набор инструментов для разработки параллельных приложений за счет стандартных классов пакета java.util.concurrent. Например, есть специальные параллельные разновидности стандартных коллекций JCF, которые не требуют блокировок, что позволяет в большинстве случаев избежать их возникновения при одновременном обращении к ним из нескольких потоков [19].

Для реализации очередей *inputQueue*, *waitQueue* и очередей к виртуальным ресурсам *virtualQueue* используется библиотека классов java.util.concurrent.ConcurrentLinkedQueue, в которой есть реализация интерфейса java.util.Queue, обеспечивающая возможность одновременного добавления и извлечения элементов нескольким потокам без необходимости использовать блокировки. Подобная структура данных для организации очереди заданий наилучшим образом подходит для входной очереди планировщика, в которую могут с одной стороны добавляться задания многими потоками, а с другой – также извлекаться и обрабатываться многими потоками пула.

3.1 Модели организации поточной обработки

Работа многих серверных приложений связана с выполнением большого количества коротких задач, поступающих от какого-либо удаленного источника. Запрос поступает на сервер посредством сетевых протоколов, через очередь JMS, путем опроса базы данных и др. Часто возникает ситуация, когда обработка отдельной задачи кратковременна, а количество запросов велико.

Одной из наиболее простых моделей является создание нового потока для каждого поступившего на сервер запроса. Данный подход носит название «поток-на-запрос» и подходит для разработки прототипа серверного приложения. Однако он имеет значительные недостатки, наиболее значимым из которых является то, что системные издержки по созданию и удалению нового потока для каждого запроса перекрывают по времени непосредственную обработку запросов.

В дополнение к издержкам создания и разрушения потоков, активные потоки потребляют системные ресурсы. Создание слишком большого количества потоков в одной JVM может привести к нехватке системной памяти или замедлению из-за чрезмерного потребления памяти. Следовательно, серверным приложениям необходимы ограничения на количество запросов, обрабатываемых в заданное время.

Решение проблемы издержек жизненного цикла потока предполагает использование потока пулов. При многократном использовании потоков издержки по их созданию распределяются на несколько задач, устраняется задержка по времени, которая раннее была необходима для создания потока. Таким образом, запрос обрабатывается немедленно, это делает приложение более эффективным. Правильно настроив количество потоков в пуле потоков, можно предотвратить пробуксовку ресурсов, заставив любые запросы, если их количество выходит за определенные пределы, ждать до тех пор, пока поток не станет доступным, чтобы его обработать [20].

Пулы потока – не единственный способ использовать множественные потоки в серверном приложении. Иногда довольно разумно генерировать новый поток для каждой новой задачи. Однако, если частота создания задач высока, а их средняя продолжительность низка, порождение нового потока для каждой задачи приведет к проблемам с производительностью.

Другая распространенная модель организации поточной обработки – наличие единого фонового потока и очереди задач для задач определенного типа. Набор инструментальных средств для абстрактных окон AWT и Swing используют эту модель, в которой есть специальный поток событий GUI, выполняющий работу по изменению пользовательского интерфейса. Однако, поскольку существует только один AWT-поток, нежелательно выполнять задачи в нем. Это означает, что в приложениях Swing часто требуются дополнительные потоки «исполнителя» для решения долгосрочных, связанных с пользовательским интерфейсом задач.

Подходы «поток-на-задачу» и «единый фоновый поток» могут довольно хорошо функционировать в определенных ситуациях. Например, подход «поток-на-задачу» хорошо работает с небольшим количеством долгосрочных задач. Подход «единый фоновый поток» функционирует эффективно при условии, что не важна предсказуемость распределения, как в случае низкоприоритетных фоновых задач. Однако большая часть серверных приложений ориентирована на обработку большого количества краткосрочных задач, и им необходим механизм для эффективного выполнения задач с небольшими издержками, а также мера управления ресурсами и предсказуемостью времени выполнения.

Пул потоков является мощным механизмом для структурирования многопоточных приложений, однако приложения, построенные с его помощью, подвержены ошибкам синхронизации и взаимоблокировок, имеют зависимую от пулов взаимоблокировку, пробуксовку ресурсов и рассеяние потока.

Исходя из вышесказанного, при реализации архитектуры брокера ресурсов подсистема планировщика была выделена в отдельный фоновый поток, который выполняется в бесконечном цикле. Это позволило отделить основную работу брокера ресурсов от вспомогательной работы планировщика по извлечению заданий из входной очереди *inputQueue* и постановку их в очереди *virtualQueue* и *waitQueue*. В основном потоке грид-сервиса CAEBeans Broker производится перераспределение виртуальных ресурсов и работа сервиса в соответсвующем контексте.

3.2 Создание масштабируемых приложений с использованием Jetty

Контейнер WSRFLite включает в себя в качестве веб-сервера Jetty. Последний предоставляет средства для параллельного выполнения нескольких запросов к грид-сервисам, что позволяет наряду с использованием поточной обработки в реализации грид-сервиса улучшить эффективность работы грид-сервиса и сократить время обработки запросов.

Веб-сервер Jetty 6 поддерживает большое количество одновременных соединений. В нем используются не блокирующие библиотеки ввода/вывода языка программирования Java (java.nio) и оптимизированная архитектура выходного буфера. Кроме того, Jetty умеет работать с долговременными соединениями – функциональная возможность, известная под названием Continuations [21]. Для использования Continuations, Jetty должен быть настроен на обработку запросов с использованием коннектора SelectChannelConnector, который позволяет сохранять соединения открытыми. Это дает возможность использовать один поток для обслуживания нескольких запросов.

Заключение

В данной статье рассмотрены архитектура и аспекты реализации компонента CAEBeans Broker. Приведено описание логической структуры, архитектуры очередей и алгоритма планирования CAEBeans Broker. Предложен подход к реализации очередей и организации потоков грид-сервиса CAEBeans Broker, а также созданию масштабируемых приложений в контейнере WSRFLite.

В качестве дальнейших направлений работы можно выделить следующие: проведение вычислительных экспериментов для оценки эффективности алгоритма планирования, основанного на очередях брокера ресурсов; реализация грид-сервиса CAEBeans Broker с использованием функциональной возможности Continuations и проведение исследования масштабируемости приложения CAEBeans Broker.

Литература

1. Дорохов В.А. Разработка виртуального испытательного грид-стенда для исследования эффекта овализации труб при термической обработке // Параллельные вычислительные технологии (ПаВТ'2009): Труды международной научной конференции (Нижний Новгород, 30 марта – 3 апреля 2009 г.). Челябинск: Изд-во ЮУрГУ, 2009. С. 457-462.
2. Радченко Г.И. Технология построения проблемно-ориентированных иерархических оболочек над инженерными пакетами в грид-средах // Системы управления и информационные технологии. 2008. № 4(34). С. 57-61.
3. Радченко Г.И., Соколинский Л.Б. Технология построения виртуальных испытательных стендов в распределенных вычислительных средах // Научно-технический вестник Санкт-Петербургского государственного университета информационных технологий, механики и оптики. № 54. 2008. С. 134-139.
4. Дорошенко А.Е., Алистратов О.В., Тырчак Ю.М., Розенблат А.П. Системы GRID-вычислений — перспектива для научных исследований // Проблемы программирования. 2005. № 1. С. 14-38.
5. Проект Condor [http://www.cs.wisc.edu/condor].
6. Portable Batch System [http://www.openpbs.org].
7. Sun Grid Engine [http://www.sun.com/software/gridware].
8. Platform Computing [http://www.platform.com].
9. Baker B.S., Coffman E.J., Rivest R.L. Orthogonal packings in two dimensions, SIAM J. Computing, 1980, v. 9, pp. 846-855.
10. Coffman E.J., Garey M.R., Johnson D.S. Tarjan R.E. Performance bounds for level-oriented two-dimensional packing algorithms, SIAM J. Computing, 1980, v. 9, pp. 808-826.
11. The Global Grid Forum [http://www.gridforum.org].
12. Foster, C. Kesselman, editors. The Grid: Blueprint for a future computing infrastructure, Morgan Kaufmann, San Fransisco, 1999.
13. Czajkowski K., Fitzgerald S., Foster I., Kesselman C., Grid Information Services for Distributed Resource Sharing, 10th IEEE International Symposium on High-Performance Distributed Computing, 2001.
14. Drozdowski M. Scheduling multiprocessor tasks - an overview, European J. of Oper. Research, 1996, v. 94, pp. 215-230.
15. Orlando S., Palmerini P., Perego R., Silvestri F. Scheduling high performance data mining tasks on a data Grid environment, Euro-Par 2002, LNCS 2400, Springer-Ferlag Berlin Heidelberg 2002, pp. 375-384.
16. Радченко Г.И. Грид-система CAEBeans: интеграция ресурсов инженерных пакетов в распределенные вычислительные среды // Параллельные вычислительные технологии (ПаВТ'2009): Труды международной научной конференции (Нижний Новгород, 30 марта – 3 апреля 2009 г.). Челябинск: Изд-во ЮУрГУ, 2009. С. 281-292.

17. Шамакина А.В. Организация брокера ресурсов в системе CAEBeans // Вестник Южно-Уральского государственного университета. Серия «Математическое моделирование и программирование». 2008. № 27(127). Вып. 2. С. 110-116.
18. Шамакина А.В. Протокол взаимодействия с брокером ресурсов в системе CAEBeans // Научный сервис в сети Интернет: масштабируемость, параллельность, эффективность: Труды Всероссийск. науч. конф. (21-26 сентября 2009 г., Новороссийск). М.: Изд-во МГУ, 2009. С. 400-402.
19. Лобанов И., Concurrent Data Types [http://javatoday.ru/2009/05/concurrent-data-types/#more-221].
20. Брайан Г. Теория и практика Java: Пулы потоков и очередь действий [http://www.interface.ru/home.asp?artId=9759].
21. McCarthy P., Ajax для Java-разработчиков: Создание масштабируемых Comet-приложений с использованием Jetty и Direct Web Remoting [http://www.ibm.com/developerworks/ru/library/j-jettydwr/index.html].