СЕКЦИЯ 1. ПАРАЛЛЕЛЬНАЯ И РАСПРЕДЕЛЕННАЯ ОБРАБОТКА ДАННЫХ

МОДЕЛИ РАСПРЕДЕЛЕННОЙ РАЗДЕЛЯЕМОЙ ПАМЯТИ

М.К. Буза, Л.Ф. Зимянин
Беларусь, г. Минск

Введение.

Кластерные архитектуры являются эффективным средством решения фундаментальных научных и инженерных задач, характеризующихся большим объемов вычислений. Кластер можно определить как пару *(N, IF)*, где *N*-набор узлов, каждый из которых содержит один или более процессорных элементов, *IF* (*Interconnec Fabric*) – фабрика межсоединений, обеспечивающая объединение узлов из *N* в некоторую сеть.

Разработка эффективных распределенных приложений для таких архитектур является достаточно сложной проблемой. К настоящему времени предложено много различных подходов к разработке параллельных программ, создано множество языков и инструментальных средств параллельного программирования. Однако существенным переломом в области разработки параллельных программ и их переносимости явилась стандартизированная технология MPI, использующая модель передачи сообщений в качестве модели выполнения. Интерфейс MPI является достаточно громоздким и сложным для прикладного программиста.

В связи с этим, для упрощения проектирования распределенных приложений в работе предлагается гибридная модель параллелизма, сочетающая модель выполнения MPI, расширенную средствами работы с общей памятью, реализованной на основе концепции распределенной разделяемой памяти *DSM (Distributed Shared Memory)* /1/. В рамках рассматриваемой модели общая память представляет общую виртуальную память кластера или сети и является надежным хранилищем данных. Практически позволяет отказаться при разработке приложений от работы с файлами. Пользовать оперирует только с объектами, создаваемыми и хранимыми в виртуальном адресном пространстве. Кроме того, предложенная модель общей памяти может быть использована в качестве платформы для построения высокоуровневых моделей распределенной обработки.

Модели распределенной разделяемой памяти. Основным понятием модели памяти кластера является хранилище данных *F*. Возможны следующие реализации хранилища: на основе сетевых файлов, вирту-

альной памяти или полиморфное – составное хранилище, которое представляет собой оболочку над коллекцией других хранилищ. В представленной версии модели памяти используются главным образом сетевые файлы.

Ниже рассмотрим две модели построения общей памяти кластерных архитектур.

1. Модель с централизованным управлением.

Виртуальное адресное пространство (ВАП) кластера рассматривается как набор $R = \{R_k\ (m_k, a_k, s_k)\}$, элементы которого будем называть регионами, $k \in I$-конечное множество (R-память). *Регион* – именованная непрерывная область памяти, разделяемая всеми узлами кластера. Образом региона R_k является некоторый участок памяти хранилища F, т.е $\exists$ Fj, в котором региону поставлено в соответствие часть файла. Каждый регион характеризуется набором атрибутов: m_k– собственно параметры памяти региона (объем, отображение на Fj, дата создания), a_k – параметры режима доступа к памяти региона (доступ по чтению/записи, блокировка доступа), s_k– параметры безопасности, связанные с ограничением прав доступа пользователей кластера. Каждый регион в пределах кластера должен иметь уникальный идентификатор. Регионы могут быть динамические и статические. Динамические регионы создаются пользователем по запросам. Статические регионы создаются администратором кластера и, в отличие от динамических, не могут быть модифицированы пользователями.

В качестве модели доступа положена классическая модель «читатели/писатели», т.е. множественный доступ по чтению для любого числа пользователей, по записи – только одному. Поэтому модель общей памяти требует поддержки строгой либо последовательной консистентности. В рамках модели реализована поддержка строгой консистентности памяти на основе внедрения в кластер специального процесса-арбитра (*referee*), обеспечивающего контроль над доступом пользователей к регионам.

Реализация модель выполнена для кластерных архитектур (высокоскоростных сегментов локальной сети) в операционной среде *Windows NT/2000/XP*, в которой сетевое взаимодействие реализуется на базе стека протоколов *TCP/IP*, а в качестве хранилища данных F используются сетевые разделяемые файлы /2/.

Центральное место в системе играет *сервер*, исполняющий роль *арбитра* (*referee*), обслуживающего запросы пользователей. Память под регионы выделяется блоками по 64 Кбайт.

Регионы представляются как объект класса *CRegion:*

```
class Cregion {
  ulong dwId;          // Идентификатор региона
  ulong dwOffset;      // Смещение от начала файла
  ulong dwBlocks;      // Количество блоков
  typedef list <CompAccess*> CmpList;
  typedef CmpList::iterator CompListIt;
  CmpList CompList;    // список компьютеров, использующих регион и атрибуты доступа
  bool Blocked;        // флаг для блокировки при записи
  attribute AccessAttr;  // атрибут возможного доступа к региону
  ...};
```

Концепция модели позволяет при создании регионов ставить им в соответствие образы в *F* некоторым оптимальным образом. В данной реализации сервер в первую очередь пытается построить его образ в файле того узла, на котором функционирует процесс, сделавший запрос. Если файл из *F* на данном узле не предусмотрен, то выбирается файл узла с минимальным удалением от данного (имеет смысл в кластере со сложной топологией). Кроме того, сервер пытается равномерно распределить создаваемые регионы по всем файлам из *F*. Это решение носит эвристический характер, поэтому в дальнейшем предполагается применение адаптивных схем создания регионов, ориентированных на определенные классы задач.

Для использования общей памяти в приложениях необходимо инсталлировать сервер (*referee*) на одном из узлов кластера. Для создания хранилища *F* используется помощник *referee* (*Sponsor*). *Sponsor* может быть запущен на любом узле кластера и с его помощью администратор создает конкретные сетевые файлы, составляющие хранилище *F*. В каждой момент времени *referee* отображает состояние общей памяти – список файлов хранилища *F*, список используемых регионов и их атрибуты, список пользователей, работающих с памятью.

Для работы с регионами пользователь должен выполнить процедуру регистрации на сервере – функция *regioninitialization*. *IP*-адреса сервера является параметром настройки среды пользователей. Связь пользователей с сервером реализуется на базе протокола *TCP*. *TCP*-соединение остается открытым до завершения работы пользователя (функция *regionfinalization)* и используется для посылки запросов пользователей за разрешением на доступ к регионам.

Для использования в приложении функций работы с регионами необходимо подключить библиотеку *rclient.dll*, включающую функции *regionmap* и *regionunmap*, которые реализуют проекцию региона (отобра-

жение) в локальное адресное пространство пользователя и его освобождение. Отображение данных региона реализовано на основе концепции ОС Windows – проекции файлов в адресное пространство процесса. Функция *regionmap* возвращает пользователю указатель на область памяти процесса, в которую спроецирован требуемый регион. Функция *regionunmap* освобождает создаваемые в процессе проецирования объекты. В случае модификации региона обеспечивается актуализация его образа.

2. Модель с децентрализованным управлением.

В отличие от предыдущей модели, ВАП кластера рассматривается как множество регионов $R = \{R_k, k=1..N\}$ фиксированного размера (*64 Кбайт*) и имеющих свой номер из фиксированного диапазона *1..N*. ВАП ставиться в соответствие распределенное хранилище, состоящее из физических страниц, причем их природа может быть любой. В программной реализации модели используются в основном физические страницы как блоки сетевых файлов. Хранилище узла жестко закреплено за определенным диапазоном номеров регионов. Механизм закрепления основан на использовании конфигурационного файла, который считывается при запуске службы управления ВАП, инсталлированном на каждом узле кластера. Конфигурационный файл задается в формате *XML* и содержит диапазон ВАП, описание физического хранилища данных, соответствующего этому диапазону, а, кроме того, сведения о других службах, на которые можно распределить нагрузку и список служб (узлов), которые имеют доступ к ресурсам этого узла.

Конфигурационный файл задает первоначальное расположение физических страниц, которые в дальнейшем могут мигрировать на другие узлы. Физическая страница, присутствующая в хранилище узла, называется резидентной. Служба узла может получить прямой доступ только к резидентным страницам. Резидентная страница может быть доступной только для чтения или для чтения и записи.

Создаваемые в ВАП разделяемые объекты приложения отображается в адрес в разделяемой области памяти и, следовательно, имеет один и тот же адрес на всех узлах. Если пользователю необходимо обратиться к разделяемому объекту, служба узла должна получить прямой доступ к соответствующим страницам. Если их нет (всех или только некоторых) в данном хранилище, то делается запрос на миграцию недостающих станиц. Узлы, на которых были страницы, помечают их как нерезидентные и отправляют их запросившему узлу, который разрешает к ним доступ.

Общая схема службы управления ВАП узла представлена на рисунке.

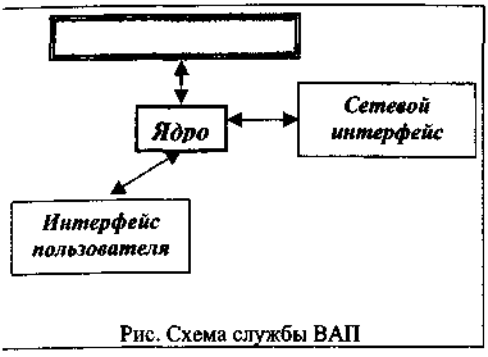

Рис. Схема службы ВАП

Основу службы составляет *ядро*, которое отвечают за управление страницами физической памяти, регионами ВАП, а так же ставят регионам в соответствие наборы страниц физической памяти и управляют состоянием семантических объектов.

Хранилище – это компонент, который обеспечивает взаимодействие с различными накопителями информации. Основная его задача – предоставить ядру унифицированный интерфейс к различным типам ресурсов.

Сетевой интерфейс реализует протокол миграции страниц, а также внутренние протоколы обмена служебной информацией.

Протокол миграции страниц обеспечивает консистентность страниц ВАП, поскольку всегда существует только одна копия страницы. Данный протокол обеспечивает высокую надежность ВАП. Однако для некоторых приложений наблюдается снижение производительности, если страницы памяти необходимы одновременно нескольким пользователям. В связи с этим, в стадии реализации находится протокол денонсирующей записи, обеспечивающий дублирование страниц при чтении.

В данной версии сетевой интерфейс реализован с использованием транспортных протоколов стека *TCP/IP*. Как показывают исследования /3/, более эффективным является применение транспортных средств, построенных на основе протоколов прямого управления каналами связи.

Интерфейс пользователя интерфейс реализован как библиотека функций *fdsm.dll*, обеспечивающих работу с ВАП. Пользовательский интерфейс поддерживает механизмы управления памятью максимально приближенные к интерфейсу аналогичных механизмов языков С и С++.

Библиотека включает следующие функции:

*void *fdsm_connect()* – обеспечивает идентификацию пользовательской программы в службе ВАП. Вызывается один раз непосредственно перед работой с библиотекой и возвращает указатель на созданный объект *Connection*. Соответственно, функция *void fdsm_disconnect(void *conn)* завершает работу с библиотекой;

*MemID fdsm_alloc(void *conn, bigint size)* – основная функция, предназначенная для выделения памяти под создаваемый объект. Результатом работы является уникальный идентификатор *MemID*,

работы является уникальный идентификатор *MemID*, который представляет распределенный объект в кластерной архитектуре. Соответственно, функция *void fdsm_free(void *conn, const MemID &obj)* освобождает память под созданным объектом (уничтожает объект);

*void *fdsm_lock(void *conn, const MemID &obj)* – возвращает пользователю указатель на память в адресном пространстве пользователя, в которое проецируется распределенный объект *MemID*. Теперь требуемый объект доступен пользователю и с ним можно работать как любым объектом C/C++. Функция *void fdsm_unlock(void *conn, void *pObj)* делает недоступным операции по изменению объекта, освобождая все ресурсы, используемые для организации доступа к распределенному объекту ВАП.

Заключение. Предложенная в работе гибридная модель существенно упрощает проектирование распределенных приложений за счет работы с разделяемыми объектами, при том, что параллельно можно использовать все возможности MPI. Как показывают эксперименты, гибридная модель наиболее эффективна при работе с объектами большого объема. Тестирование работы с большими регионами (10-15 Мбайт) в сети *Fast Ethernet* (100 Мбит/с) показало достижение реальной скорости транспорта данных между узлами кластера до 70-75 Мбит/с.

Литература

1. Amza, C, A. L. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu, and W. Zwaenepoel. 1996. Treadmarks: Shared memory computing on networks of workstations. *IEEE Computer* 29, 2 (February): 18-28.

2. Буза М.К., Зимянин Л.Ф. Модель общей памяти для кластерных архитектур // Международная научн. Конф. «Суперкомпьютерные системы и их применение» SSA'2004: Доклады конф. – Мн.: ОИПИ, 2004. – С.77-80.

3. Буза М.К., Зимянин Л.Ф. Разработка и программная реализация эффективных транспортных средств для кластерных архитектур //Информационные системы и технологии (IST'2004): Материалы II Междунар. Конф.: В 2 ч. Ч. 2. – Мн.: Академия управления при Президенте Республики Беларусь, 2004. – С. 90-95.