УДК 004.43

С.И.БОГУЧАРСКИЙ, Н.И.САМОЙЛЕНКО, д-р техн. наук
Харьковский национальный университет городского хозяйства имени А.Н.Бекетова

РЕАЛИЗАЦИЯ ЗАДАЧ РАСПРЕДЕЛЕННЫХ СИСТЕМ НА ПЛАТФОРМЕ .NET

Рассматривается разработка программ для распределенных систем на платформе .Net для операционной системы Windows. Проводится сравнительная характеристика инструментов для Unix-платформ. Предложена модель реализации программ для распределенных систем на платформе .Net с использованием языка программирования MC#.

Розглядається розробка програм для розподілених систем на платформі .Net для операційної системи Windows. Проведена порівняльна характеристика інструментів для Unix-платформ. Запропонована модель реалізації програм для розподілених систем на платформі .Net з використанням мови програмування MC#.

We consider the development of programs for distributed systems on the platform .Net for the operating system Windows. Conducted comparative analysis tools for Unix platforms. A model of the implementation of programs for distributed systems on the platform .Net using the programing in MC# language.

Ключевые слова: параллельное программирование, распределённое программирование, MC#, вычислительная система

С каждым годом IT-специалистам становится всё сложнее и сложнее «подтверждать» закон Мура (частота процессоров удваивается каждые два года). Одним из альтернативных способов обеспечения требуемого быстродействия вычислительных систем является использование многоядерных процессоров, кластеров и GRID-сетей, то есть параллельного нераспределенного программирования.

Процесс написания параллельных программ намного сложнее процесса написания последовательных программ. Помимо того, что программисту приходится думать о логике, размещении (т.е. разбиении на библиотеки) и архитектуре кода, ему еще необходимо [1]:

1) моделировать способы синхронизации параллельно исполняемых участков кода;

2) сличать соответствие результатов работы программы с результатами работы аналогичной последовательной программы;

3) не допускать возникновения санд-эффектов (выполнение кода в параллельном режиме с другим кодом, которое может привести к неожиданным побочным эффектам).

Кроме того, зачастую многие программы вычислительного характера вообще не распараллеливаются (в силу специфики алгоритмов), то есть наблюдается обратный эффект снижения производительности при переходе от одного процессора к нескольким. Как правило,

подобный эффект наблюдается при переходе от n процессоров к $n + 1$, но может иметь место и при $n = 1$.

Ситуация ещё более усложняется при переходе от кластеров к метакластерам и GRID-сетям, так как программисту приходится учитывать такие факторы как:

1) большая латентность между узлами/кластерами, при которой производительность различных элементов вычислительной сети (ВС) может отличаться на несколько порядков;

2) потеря соединения с некоторыми из элементов ВС;

3) потеря отдельных элементов ВС и др.

В течение нескольких последних десятилетий было создано немало систем и специальных библиотек, помогающих в написании параллельных распределённых программ. Многие из них написаны на языках программирования C и Fortran или же реализованы как надстройки над этими языками. В языке C основным стандартом для написания параллельных программ является MPI (Message Passing Interface). Наряду с многочисленными достоинствами язык C и библиотеки типа MPI имеет место и ряд недостатков. К последним следует отнести:

1) низкий алгоритмический уровень языка;

2) сложность обучения;

3) наличие большого количества программных ошибок, вызванных отсутствием проверок выхода за пределы массива на синтаксическом уровне, автоматической сборкой мусора и т.п.;

4) отсутствие возможности автоматического или полуавтоматического распараллеливания;

5) отсутствие возможности автоматического переноса сложноустроенных объектов с одного узла на другой, то есть отсутствие унифицированных механизмов сериализации/дисериализации;

6) несовместимость библиотек типа MPI и надстроек над MPI, разрабатываемых разными пользователями и рассчитанных на разную архитектуру вычислительных систем.

Изложенные недостатки привели к идее создания нового языка программирования, который соответствовал бы следующим требованиям [2]:

1) при использовании такого языка программист должен полностью абстрагироваться от среды исполнения, то есть неважно, где будет исполняться программа (на одном узле, кластере, метакластере или GRID-сети);

2) язык должен быть высокоуровневым с поддержкой автоматической сборки мусора, обширными встроенными библиотеками и удобным синтаксисом (например, надстройкой над Java или C#);

3) различные конфигурации (узел/кластер/метакластер/GRID-сеть) могут поддерживаться различными средами исполнения, но синтаксис языка должен оставаться неизменным для всех конфигураций;

4) система исполнения параллельных программ должна сама распределять нагрузку по узлам вычислительной сети без участия программиста;

5) система исполнения также должна автоматически поддерживать передачу по сети сложноустроенных в памяти объектов, при этом способ передачи должен быть протокольно определен (протокол передачи данных – TCP/IP, MPI и др.);

6) язык должен поддерживать синтаксические конструкции для синхронизации параллельных/распределённых нитей, быть лёгким для понимания и обучения.

Язык программирования MC# удовлетворяет выше перечисленным требования. Изначально все системы исполнения для языка MC# создавались для систем типа Unix/Linux на базе платформы Mono. Несмотря на то, что почти все кластеры сегодня сделаны на базе Linux/Unix-платформ, большинство C#-программистов используют Windows и стандартный компилятор от Microsoft. Чтобы привлечь больше пользователей к работе с кластерами, была разработана специальная кластерная версия системы исполнения MC# под Windows [3].

Архитектура системы исполнения под Windows немного отличается от аналогичной системы для Linux/Unix, однако синтаксис языка является одинаковым для обеих систем, то есть можно разрабатывать и отлаживать параллельную программу на своей локальной Windows-машине. После отладки её можно скопировать на Linux-кластер и запустить с помощью специализированной системы исполнения для Linux. Такая система состоит из трёх инсталляционных пакетов:

1) MC#.Cluster – Compiler & Utils;

2) MC#.Cluster – Resource Manager;

3) MC#.Cluster – Work Node.

С помощью перечисленных пакетов можно довольно легко «развернуть» кластер на обычных домашних компьютерах, предварительно установив .Net. Для этого сначала необходимо выбрать «фронтенд» кластера, то есть главный узел кластера, с которого будут запускаться все пользовательские программы. Следует заметить, что

возможен вариант исполнения MC#-программ в локальном режиме, когда перемещаемые функции будут выполняться в отдельных потоках. В этом случае достаточно установить первый пакет MC#.Cluster – Compiler & Utils. Также можно «эмулировать» кластер на одной машине, установив на ней сразу три пакета одновременно. Это позволит минимизировать затраты необходимые на отладку программ, так как на этапе отладки не нужны ресурсы инсталляционных кластеров.

Первый пакет Compiler & Utils содержит в себе компилятор, документацию и инструментарий для управления состоянием кластера. Его достаточно установить только на главном узле кластера (фронтенде).

Второй пакет Resource Manager содержит в себе специальный менеджер распределения ресурсов, реализованный в виде Windows-сервиса. Этот сервис запускается при запуске утилиты mcsboot, входящей в комплект первого инсталляционного пакета. Данный сервис следит за распределением нагрузки на узлах кластера. Этот пакет также должен быть установлен на фронтенде.

Третий пакет Work Node содержит в себе специальный Windows-сервис, который принимает задания от менеджера распределения ресурсов. Он также загружается автоматически после инсталляции или загрузки системы.

Система была разработана с помощью Microsoft Visual Studio полностью на языке C# за исключением компилятора, который изначально был написан на языке Java для Linux-платформы и был конвертирован в .Net-сборки с помощью IKVM. Общая схема кластера и расположения программных элементов изображена на рис. 1.

В отличие системы исполнения для Linux-версии, для Windows-версии не нужна поддержка протокола SSH, так как все необходимые сервисы запускаются автоматически при запуске системы. Однако при выполнении программы на нескольких узлах необходимо, чтобы исполняемые модули (.exe и .dll файлы) лежали в одной и той же папке на всех узлах кластера. Этого можно достичь двумя способами: либо перед запуском просто скопировать дистрибутив программы на узлы «вручную» или настроить сетевую папку, которая должна быть доступной на всех узлах.

Система исполнения, так же как и в Linux-версии, поддерживает одновременное выполнение нескольких пользовательских задач, то есть она является мультизадачной. Все запущенные процессы на узлах исполняются в контексте Windows-сервисов Resource Manager и Work Node, то есть имеют одинаковые права для всех пользователей, для которых запущен сервис Work Node.

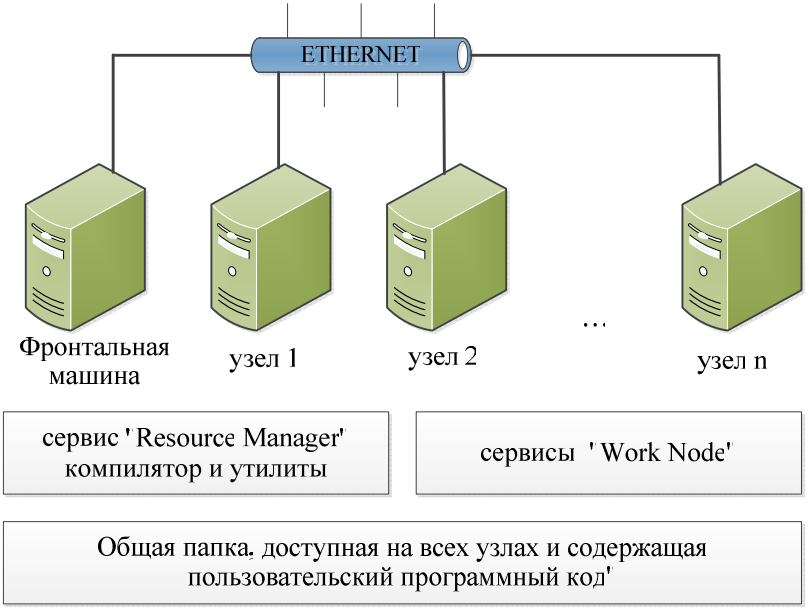

Рис. 1 – Общая схема кластера и расположение программных элементов

Распределение нагрузки по узлам производится менеджером распределения ресурсов централизованно «по кругу» для каждого из запущенных приложений, то есть при запуске приложения 1 на узлах a, b, c, d именно эти узлы и будут использоваться по очереди для отработки перемещаемых функций в рамках этого приложения. Запуск пользовательских приложений ничем не отличается от запуска приложений в Linux-версии. Распределение запущенных приложений изображено на рис. 2.

На базовом уровне версия системы исполнения для Windows-платформ в большинстве своём повторяет архитектуру версии MC# для Linux-платформ за исключением использования Windows-сервисы вместо «процессов-демонов» и отсутствия необходимости в SSH. Кроме того, Windows версия реализована в виде трёх отдельных инсталляционных пакетов, которые необходимо устанавливать только на тех узлах, где они действительно нужны.

Все компоненты системы, будь то менеджер распределения ресурсов, «рабочий узел», или клиентская программа, запущенная на

фронтальной машине или любой другой машине кластера, обмениваются между собой сообщениями по специальному протоколу (в текущей реализации не используется библиотека System.Remoting). Протокол взаимодействия между компонентами системы в основном аналогичен Linux-версии (прикладной протокол поверх TCP/IP) – все сообщения в системе состоят из заголовка и тела сообщения. Заголовок сообщения может содержать несколько строк, разделённых между собой символом перевода строки «\n» Первая строка заголовка должна начинаться с уникального слова (идентификатора команды), по которому принимающая сторона может распознать поступившую команду, например, это может быть «stop», «sessioninitialized, «hal» и т.п. Тело сообщения может отсутствовать или же наоборот содержать какие-либо большие бинарные данные, например, вызов перемещаемого метода в сериализованном виде или канальное сообщение.

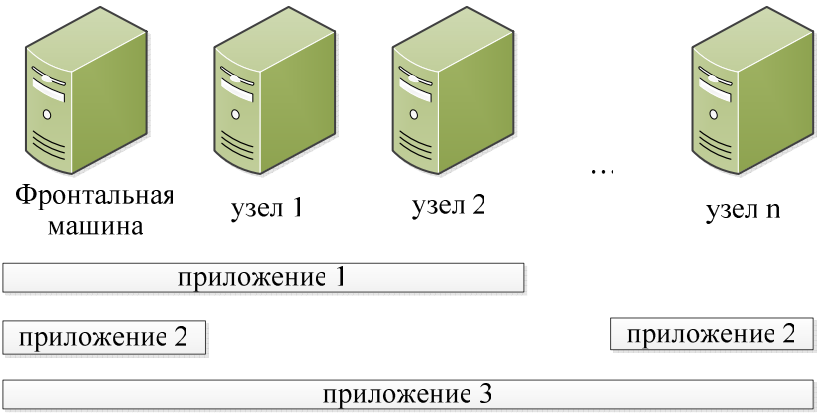

Рис. 2 – Возможное распределение запущенных приложений по узлам кластера

В случае передачи данных с одного узла на другой открывается новое соединение. После открытия соединения команда передаётся на другой узел и получает какой-либо ответ от принимающей стороны (ответ может и отсутствовать). После завершения обмена информацией соединение закрывается, чтобы как можно скорее освободить занимаемые ресурсы. Отсутствие постоянных соединений между узлами гарантирует масштабируемость системы, то есть можно легко увеличивать число узлов в кластере [4].

Основным направлением дальнейшего развития работ по проекту МС# будет создание средств разработки для Windows-платформ и их интеграция с Visual Studio .Net. В частности, планируется поддержка возможности создания МС#-проектов и создание «распределённого» отладчика. Далее, в случае запуска МС#-программ в локальном режиме на многоядерной машине, текущая версия Runtime-системы опирается на операционную систему для распределения задач по процессорам. Однако можно достичь большей производительности, если Runtime-система сама будет распределять movable-методы по имеющимся процессорам.

Таким образом рассмотрены предпосылки создания языка программирования МС#, а также представлена новая кластерная система исполнения для этого языка для Windows-платформ. Также рассмотрены отличия этой системы от соответствующей версии для Linux-платформ и определены основные направления дальнейшего развития системы. Внедрение языка МС# обеспечивает разработку распределённых систем не только на системах Unix-платформ, а так же и на Windows.Net-платформах.

1. Таненбаум Э. Распределенные системы. Принципы и парадигмы [Текст] / Э. Таненбаум, М. Ван Стеен. – Спб.: Питер, 2003. – 877 с.

2. Сердюк Ю.П. Введение в параллельное программирование на языке МС# [Текст] / Ю. П. Сердюк. – Переславль-Залесский.: ИПС РАН, 2007. – 51 с.

3. Язык программирования МС#. Докуметация: [Электронный ресурс]. – Режим доступа: URL : http://www.mcsharp.net/ – Inroduction to programming in MC# language, in Russian (pdf)

4. Гергель В.П. Высокопроизводительные вычисления для многоядерных многопроцессорных систем [Текст] / В.П. Гергель. – Н. Новгород.: НГУ, 2010. – 421 с.

Получено 11.10.2013