электронный журнал
МОЛОДЕЖНЫЙ НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51038.

**УДК 004.75+519.171.2**

# Иерархическая асинхронная модель представления графов в параллельных и распределенных системах

***Петров Ю.К.***, *студент*
*Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,*
*кафедра «Компьютерные системы и сети»*

*Научный руководитель: Иванова Г.С., д.т.н., профессор*
*Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана*
*bauman@bmstu.ru*

## Введение

Графы используются при решении широкого спектра задач, связанных с анализом и синтезом структур объектов. Так, графы могут быть использованы для реализации древовидных структур в поисковых запросах, семантическом анализе текстов, моделировании сложных устройств, представимых в виде множества состояний.

В обрабатывающей программе графы обычно представляют, таблично задавая отношения смежности или инцидентности вершин и/или ребер. Такой подход был эффективен для однопроцессорных вычислительных систем. В настоящее время в большинстве случаев используются многопроцессорные системы, зачастую распределённые, для которых представление таблицами смежности или инцидентности может оказаться неэффективным.

В [1-3] предлагается представление, рассчитанное на параллельную обработку. При этом:

1. Вычислительная среда предполагается гомогенной.
2. Вершины графа представляются некоторыми пассивными структурами данных.
3. Ответственность за распределение вершин графа лежит на основном (управляющем) узле вычислительной установки.

Таким образом, предлагаемому в [1-3] представлению свойственно централизованное управление структурой графа, что обуславливает синхронность исполняемых над графом операций и не позволяет реализовать алгоритмы, гарантирующие параллелизм на всех стадиях обработки данных, или даже эффективно

использовать вычислительные мощности. Исходя из основного типа используемого способа межпроцессного взаимодействия, данное представление далее будем называть *синхронным*.

Для обеспечения более эффективного использования потенциала распределённых систем требуется изменить подход, распределив управление по специализированным процессам, что снимет нагрузку с основного управляющего процесса, обеспечив минимум блокировок.

Предлагаемый в настоящей работе способ представления графов ориентирован на параллельные и распределённые системы. Он основывается на том, что вершины графа представлены легковесными процессами внутри виртуальной машины (в данной работе для этого была использована нотация языка Erlang [4]). Идея, лежащая в основе данного представления, аналогична идее, заложенной в модели акторов: существует некоторое количество относительно автономных сущностей (в данном случае – вершин), которые могут действовать одновременно и обмениваться информацией друг с другом в процессе решения задачи.

**Постановка задачи**

Требуется разработать модель представления графов, удовлетворяющую следующим условиям:

1. Обеспечение высокой степени параллелизма при выполнении операций над графом.
2. Применимость к разным типам параллельных систем (SMP, NUMA…).
3. Возможность применения в гетерогенных средах.

**Основные концепции предлагаемой модели представления графа**

Предлагаемая модель подразумевает концепции, отличные от описанных выше:

1. Ответственность за распределение вершин графа распределена между различными процессами, отвечающими за первичное размещение, групповую коммуникацию, балансировку.
2. Вершины графа являются легковесными процессами внутри виртуальной машины.

На рисунке 1 изображена простейшая схема графа, представленного с помощью данной модели.

В модели используется терминология, принятая в языке программирования Erlang:

- Supervisor – процесс-наблюдатель. Он может не только выполнять вычисления, но и отвечает за отказоустойчивость порождённых им процессов, т.е. процессов, располагающихся ниже его в иерархии данной модели.
- Worker – обычный рабочий процесс, этими процессами представляются вершины графа.
- Node – представляет собой узел, в котором запущены легковесные процессы, реализующие модель. Таким образом, узлом является экземпляр виртуальной машины, т.е. на одном многоядерном процессоре или даже процессорном ядре могут работать несколько узлов.

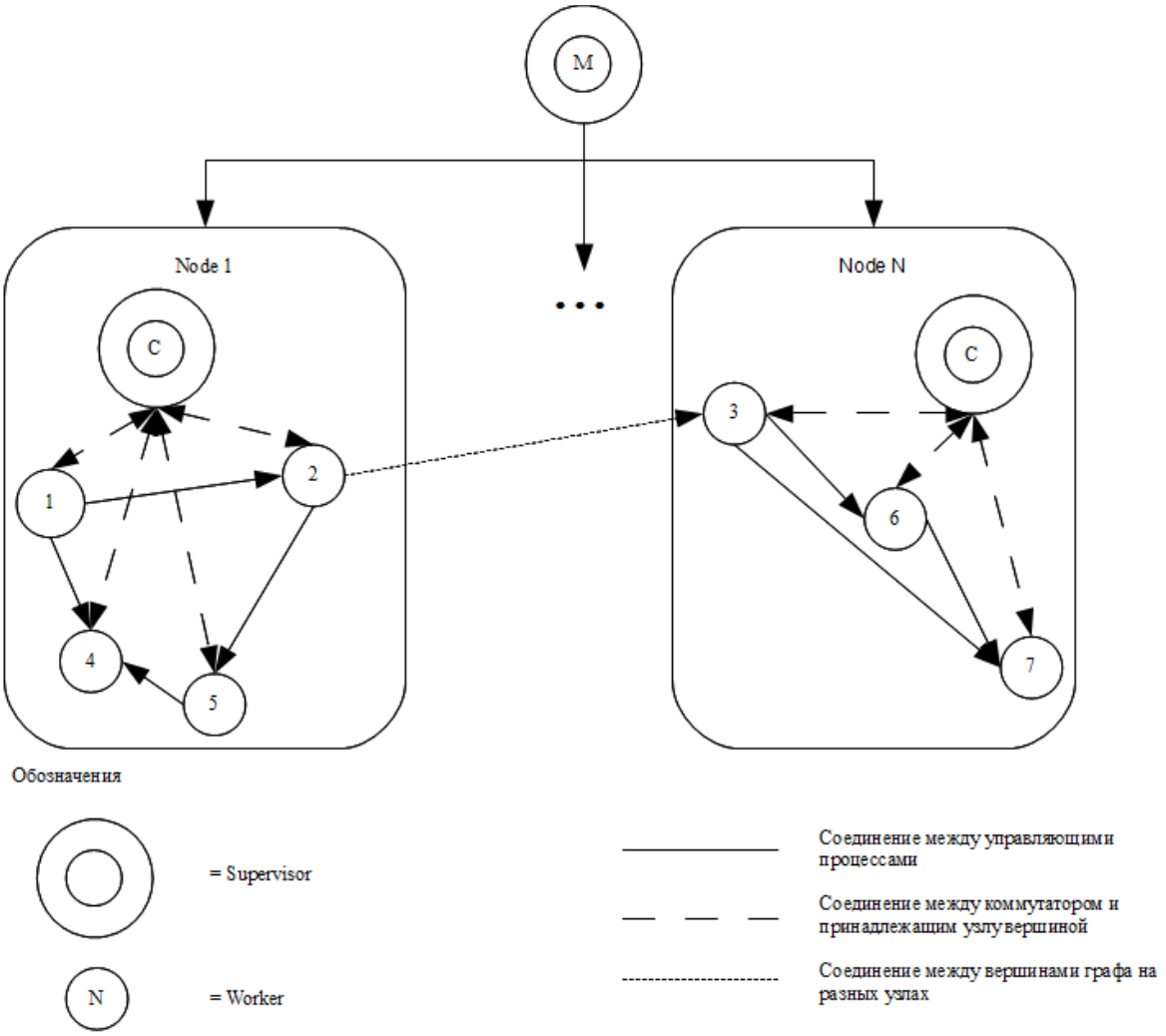


Рис. 1. Простейшая модель графа, распределённого по двум вычислительным узлам

**Иерархия процессов в модели**

Для обеспечения максимальной возможности распараллеливания организуем (реализуем) следующие типы процессов:

1. Мастер Master (M) – существует всегда только один мастер, отвечает за взаимодействие пользователя с графом, добавление вершин, первичное размещение графа (при помощи поисковых агентов – см. далее), адресацию узлов, адресацию вершин по номерам узлов. Не может быть завершён, так как является корнем иерархии.
2. Поддерживающий мастер Support Master (SM) – процессы этого типа порождаются мастером, могут существовать на различных виртуальных машинах. Выполняют те же задачи, что и мастер, за исключением первичного размещения узлов, обеспечения отказоустойчивости коммуникаторов и порождения новых поддерживающих мастеров. Может быть завершён.
3. Глобальный вспомогательный процесс Global Service Process (GSP) – относится к необязательным процессам, представляет собой абстракцию, конкретизируемую разработчиком. Вспомогательные процессы не имеют, на данный момент, конкретных требований к реализации, помимо того, что они не должны вносить синхронность в модель, если это явно не требуется. Глобальные вспомогательные процессы имеют доступ к данным мастер-узла и абстрактным глобальным данным, предоставленных мастерами, таким как ID вершин, однако по умолчанию не имеют доступа к локальным данных узлов, например, к pid вершин. Могут использовать механизм передачи сообщений для связи с другими узлами: поддерживающими мастерами, коммуникаторами, другими вспомогательными процессами. Примером глобального вспомогательного процесса может являться поисковый агент, который ищет в БД вершин связные домены и сообщает информацию о них узлам.
4. Коммуникатор Communicator (C) – существующий на каждом узле в единственном экземпляре процесс, отвечает за размещение и адресацию процессов вершин внутри узла, используется для доступа к ним. Может быть завершён только при удалении узла и только после перераспределения принадлежащих узлу вершин. Отвечает за отказоустойчивость вершин графа, в случае аварийного завершения вершины (или всего узла) коммуникатор обеспечивает восстановление данных, обращаясь за информацией к главному узлу.

5. Локальный вспомогательный процесс Local Service Process (LSP) – относится к необязательным процессам, отличается от глобального только тем, что по умолчанию имеет доступ только к локальным данным узла, предоставленных коммуникатором.
6. Вершина графа Vertex (V) – рабочий процесс вершины графа. Содержит в своём состоянии адрес коммуникатора, информацию о вершинах, указывающих на вершину и вершинах, на которые указывает вершина. Процесс вершины может быть завершён при удалении вершины, при балансировке нагрузки (в этом случае создаётся новый процесс вершины на том узле, куда переносится вершина), в случае ошибки (в этом случае коммуникатор должен предпринять действия по устранению возникшей ошибки).

Перечисленные процессы могут быть разделены на 3 уровня:

1. Глобальные управляющие процессы – процессы, взаимодействующие с графом в целом, некоторые из них могут быть использованы как интерфейсы для связи с пользователем или другими приложениями.
2. Локальные управляющие процессы – процессы, взаимодействующие только с частью графа на узле и своими вышестоящими родительскими процессами.
3. Рабочие процессы – вершины.

Полученная иерархия схематично представлена на рисунке 2.

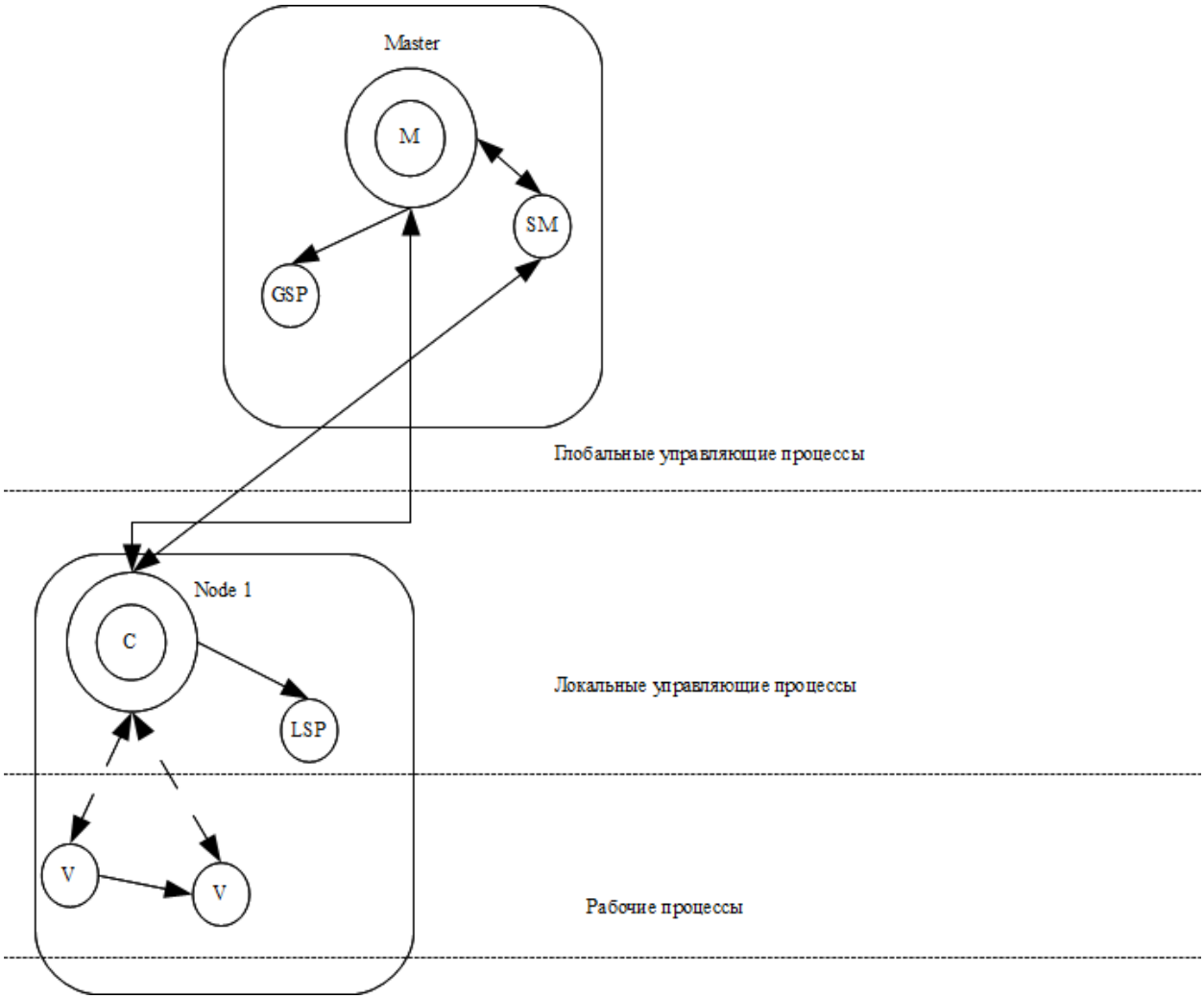


Рис. 2. Иерархия процессов модели

Мастер-узел, реализованный легковесным процессом, в рабочем состоянии содержит две таблицы. Первая таблица содержит информацию об управляемых узлах: номер главного процесса узла и идентификатор соответствующего процесса внутри конкретной виртуальной машины pid. Главным процессом узла может являться поддерживающий мастер или коммуникатор. Вторая таблица отображает распределение вершин по узлам.

Теоретически, эти таблицы могут быть сведены в одну, однако при этом усложнится обеспечение отказоустойчивости. В случае экстренного завершения узла он может быть просто перезапущен, вершины графа, размещенные на нём ранее, должны быть переразмещены, однако таблица с размещением вершин не изменится. Итого надо внести одно изменение. Если таблица будет всего одна, потребуется исправить каждую строку, где записан pid коммуникатора отказавшего узла.

Поддерживающие мастер-узлы обладают теми же данными, что и мастер-узел в модели без поддерживающих узлов. Поддерживающие мастер-узлы не имеют информации друг о друге, т.к. в этом нет необходимости. Они могут использоваться как удалённые консоли управления, добавляя, удаляя вершины графа, запуская вычисления над графом. При этом отказ одного из поддерживающих узлов никак не отражается на работоспособности: мастер-узел может просто перезапустить отказавший поддерживающий мастер-узел, благодаря асинхронной передаче сообщений отправка сообщения по недействительному адресу (например, от коммуникатора к отказавшему поддерживающему мастер-узлу) не привносит практически никаких накладных расходов.

С точки зрения мастер-узла поддерживающие мастер-узлы почти не отличаются от процессов коммуникаторов, поэтому помещены с ними в одну таблицу. Это обеспечивает быстрое восстановление в случае отказа поддерживающего мастер-узла: мастер-узел ищет pid отказавшего узла (полученный из сообщения экстренного завершения) в таблице узлов, перезапускает процесс, отправляет ему требуемые данные (в зависимости от типа узла) и обновляет информацию в таблице узлов.

Реализация коммуникатора в виде легковесного процесса предполагает всего одну таблицу, отображающую соответствие номеров вершин графа, размещенных на данном узле, pid рабочих процессов, ассоциированных с этими вершинами, количество связей с вершинами данного узла (графа Local) и других узлов (Node2...NodeN для Node1). Процесс также обладает параметром State, под которым понимается внутреннее состояние процесса: в число переменных внутреннего состояния процесса всегда входит pid мастер-узла, хранящийся в переменной типа список. Другой вариант представляет собой реализацию с помощью таблицы, однако гарантированно она будет содержать лишь одно значение, что нецелесообразно. Во внутреннее состояние коммуникатора так же входят pid поддерживающих мастер-узлов, они хранятся в том же списке, что и pid мастер-узла, но всегда после него.

Коммуникатор является ответственным за обновление информации о размещении вершин: при переразмещении вершины он отправляет соответствующую информацию мастер-узлу и всем поддерживающим мастер-узлам.

Вершина, представленная в виде процесса виртуальной машины, не обладает никакими таблицами, т.к. в модели графа с десятками тысяч вершин реализация потребляла бы слишком много памяти, вся необходимая информация содержится в

состоянии процесса вершины, описываемом параметрами ID, Pid of comm, List of neighbors, List of connections.

Под параметром ID понимается постоянный уникальный идентификатор вершины (её номер). Pid of comm — pid коммуникатора узла, которому принадлежит вершина. Под списком соседей List of neighbors понимается список вершин, представленных кортежами вида {ID, pid}, которые указывают на эту вершину. Список соединений List of connections представляется списком всех вершин, на которые указывает вершина, запись представляется кортежем вида {ID, pid, Weight}, где Weight – вес ребра.

Возможный вариант единичной записи для списков List of neighbors/List of connections: {ID, pid}. Достоинством такой записи является что:

1. Многие операции выполнимы без участия в них коммуникатора.
2. Упрощена операция переразмещения вершин в процессе балансировки нагрузки за счёт децентрализации хранения данных: в случае появления новых связей или удаления старых не требуется обращаться к глобальной базе данных, тем самым блокируя её.

Однако при этом необходимо обновление состояния при каждом изменении pid вершины из любого списка, что также следует учитывать.

В качестве механизма межпроцессного взаимодействия используем асинхронные сообщения. Это значит, что каждый процесс обладает «почтовым ящиком» (mailbox), из которого он может выбирать сообщения в порядке поступления или произвольно, по тегам, например, сообщения от определённого отправителя. Следует заметить, что при этом операции получения сообщения и его чтения могут быть разделены во времени. Базовыми операциями в реализациях существующих моделей, как правило, являются синхронные операции чтения/записи, в предлагаемой – асинхронная передача сообщения, где сообщение представляет собой запрос или ответ.

**Выводы**

Выполненный анализ позволяет сделать вывод о том, что предлагаемая асинхронная модель представления графа в обрабатывающей программе потенциально обеспечивает лучшие возможности распараллеливания операций, выполняемых над графами при решении задач структурного анализа и синтеза, что позволяет уменьшить время решения таких задач на ЭВМ. Возможно заметное ускорение балансировки нагрузки в условиях динамически изменяющейся структуры графа, а также решение ряда проблем, описанных в [3], в частности: применение модели для гетерогенных систем, метакомпьютинг.

С учетом того, что многие задачи структурного анализа и синтеза являются NP-полными, представление графов подобными асинхронными моделями представляется перспективным.

## Список литературы


1. Гергель В.П. Теория и практика параллельных вычислений. М.: Интернет-Университет Информационных Технологий; БИНОМ. Лаборатория знаний, 2013. 423 с.
2. Якобовский. М.В. Введение в параллельные методы решения задач. М.: Изд-во Московского университета, 2013. 328 с.
3. Shloegel K., Karypis G., Kumar V. Graph Partitioning for High Performance Scientific Simulations. Minneapolis: University of Minnesota, 2000. 40 p.
4. Чезарини Ф., Томпсон С. Программирование в Erlang. М.: Изд-во «ДМК», 2012. 488 с.