ВЫЯВЛЕНИЕ ТОПОЛОГИИ КОММУНИКАЦИОННОЙ СРЕДЫ ВЫЧИСЛИТЕЛЬНОГО КЛАСТЕРА ПО РЕЗУЛЬТАТАМ НАГРУЗОЧНОГО ТЕСТИРОВАНИЯ

П.С. Банников, А.Н. Сальников

Разработчик параллельной программы для вычислительного кластера заинтересован в её наибольшей производительности. Не последнюю роль в повышении производительности играет оптимизация под архитектуру конкретного кластера, то есть конкретное распределение параллельных процессов по его процессорам (или ядрам процессоров). Оптимальное распределение невозможно произвести без знания архитектуры вычислителя, то есть физической топологии его коммуникационной среды. Как правило, информацию об архитектуре даёт спецификация (или документация) вычислителя, составленная производителем. Однако распределение процессов по процессорам, опирающееся на спецификацию, может быть далеко от оптимального. Такое поведение — следствие недостатков спецификаций:

1) идеализированность представления: указана только некая идеальная топология коммуникационной среды кластера — тор, решётка и тому подобное..., а также пиковая или средняя пропускная способность каналов;

2) несоответствие действительности: сюда входят различные «аномалии», например, слишком низкая пропускная способность какого-нибудь канала, хотя было заявлено обратное, или ошибки при конструировании кластера;

3) инертность информации: состояние коммуникаций меняется быстрее, чем обновляется спецификация.

Также спецификация может просто отсутствовать, например, в случае, когда кластер собран из компьютеров, объединённых одним коммутатором.

Топологией назовём ориентированный взвешенный граф, однозначно сопоставляемый архитектуре коммуникационной среды кластера. Вершинами графа являются вычислительные узлы кластера, дугами — физические или логические связи между узлами. Здесь и далее в статье будем отождествлять понятия «топология» и «граф».

Вторым немаловажным пунктом является обнаружение неполадок в работе коммуникационной среды кластера. Любое отклонение в работе коммуникаций повлияет на автоматически выявленную топологию. По отличиям между выявленной топологией и той топологией, что указана в спецификации, можно судить о сбоях в системе и даже об ошибках проектирования кластера.

В-третьих, если спецификация отсутствует, а топология заранее не известна, появляется возможность для написания спецификации.

Итак, знание текущего состояния топологии коммуникационной среды кластера способствует оптимизации параллельных программ под архитектуру данного кластера, а также помогает обнаруживать разнообразные ошибки в работе коммуникационной среды кластера. Получение текущего состояния топологии являлось целью проделанной работы.

Для обхода недостатков спецификаций архитектуры было предложено использовать нагрузочные тесты коммуникационной среды. На факультете ВМК МГУ им. Ломоносова такие тесты были разработаны в составе пакета программ PARUS [1] (программа network_tests2). Тест заключается в обмене MPI-сообщениями различной длины между процессами и последующих замерах времени передачи. Выходными данными теста является совокупность матриц: для каждой длины сообщений выдаётся квадратная матрица, каждая ячейка которой содержит величину замера времени передачи от одного процесса другому.

В рамках статьи ставятся три основные задачи:

1) по данному набору матриц выявить топологию коммуникационной среды кластера;

Результирующий граф получается после слияния графов, соответствующих определённым длинам сообщений. Так как одни и те же дуги могут входить в совершенно различное число сливаемых графов, у каждой дуги появляется ещё один параметр: *вероятность её существования* в результирующем графе.

2) отобразить в трёхмерном пространстве получившийся граф с сохранением длин его дуг;

3) разработать формат для описания эталонной топологии вычислительного кластера (то есть топологии, полученной из спецификации вычислительного кластера), а также разработать методы сравнения эталонной топологии с топологией, получаемой в п. 1; разработать и реализовать алгоритм отображения различий в трёхмерном пространстве.

К системе тестирования и к результирующему графу предъявляется одно главное требование: как *можно точнее* (в смысле п. 3) *приблизить реальную (физическую) архитектуру кластера*.

Сначала остановимся подробнее на задаче выявления топологии.

Задача актуальна сразу в нескольких смежных областях, однако её актуальность была, по-видимому, не ясна в области вычислительных кластеров ввиду наличия спецификаций. Этим же объясняется отсутствие алгоритмов выявления топологий.

К новому алгоритму предъявим следующие *требования*: произвольная структура результирующего графа, полиномиальное (от числа MPI-процессов) время работы, корректная обработка процессов на общей памяти. На вход алгоритму должна подаваться одна из матриц, полученных в результате нагрузочного тестирования. При этом выявляется *локальная топология*. Далее набор локальных топологий сливается в одну итоговую топологию путём отождествления вершин и присваивания вероятностей существования дуг. Таким образом, *вероятность существования* дуги — отношение количества локальных топологий, содержащих эту дугу, к общему количеству локальных топологий. Полученная в итоге топология считается выявленной топологией коммуникационной среды вычислительного кластера.

В статье предложен оригинальный алгоритм, выявляющий локальную топологию и удовлетворяющий всем вышеперечисленным требованиям. Более того, для корректной работы описанного ниже алгоритма не требуется симметричность входной матрицы.

Пусть в тесте участвовало n MPI-процессов, тогда входная матрица M имеет размер nxn. Введём вспомогательные структуры:

- граф G — список из n вершин; для каждой вершины хранится список смежных с ней вершин и вес соответствующей дуги; максимальный размер G равен n2, так как полносвязный ориентированный граф содержит ровно n2 дуг;
- множество S, хранящее порядковые номера процессов на общей памяти (максимальный размер — (n-1));
- массив D индикаторов «завершённости» вершин размером n.

Объявляются 2 параметра алгоритма: εs и εd.

Параметр εs объясняется следующим образом: предположим, что имеется группа процессов-вершин на общей памяти, и пусть вершины i и j уже соединены дугой с весом w. Тогда остальные пары вершин будут соединяться дугами, только если $M_{ij} = w * (1 + \varepsilon_s)$ (1). Параметр εs определяет допустимый разброс весов дуг, соединяющих процессы-вершины на общей памяти. Рекомендуемый интервал значений для εs — от 0 до 0.5.

Параметр εd позволяет управлять типом канала, представленного парой дуг. Поскольку матрица M несимметрична, веса двух дуг, соединяющих одну и ту же пару вершин, могут сильно различаться. Если отношение значений весов таких дуг превышает 1+εd, то канал считается симплексным, а дуга с большим весом уничтожается. В противном случае канал считается дуплексным. Рекомендуемый интервал значений для εd — от 0 до 1.

Следует отметить, что наличие симплексного канала в выявленной топологии является либо свидетельством неполадок в системе, либо эффектом от работы алгоритма маршрутизации сообщений.

Работа алгоритма разбивается на 3 этапа: 1) подготовка; 2) выявление остовного дерева; 3) восстановление циклов.

На этапе 1 происходит инициализация вспомогательных структур.

Суть этапа 2 заключается в выявлении остовной структуры локальной топологии: предполагается, что локальная топология состоит из набора групп процессов-вершин на общей памяти, группы соединены цепью дуг, причём между двумя группами вершин есть максимум 1 цепь дуг. Вершины внутри группы могут быть соединены произвольным образом, но так, чтобы группа представляла собой связный подграф.

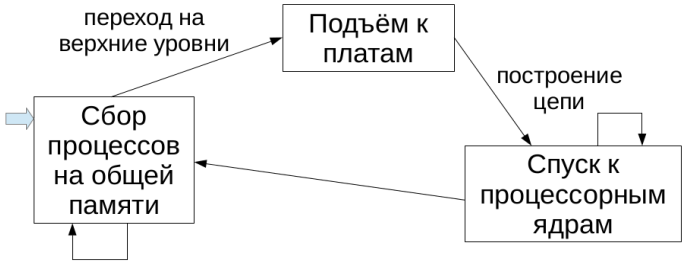

Схема 1: работа этапа 2

На схеме 1 приведён автомат, описывающий работу этапа 2. Голубой стрелкой отмечено начальное (и одновременно конечное) состояние автомата.

В начальном состоянии находится минимальное значение в M. Считается, что пара процессов-вершин, соответствующих найденному минимуму, расположена на общей памяти. Проводится сбор самой первой группы процессов на общей памяти до тех пор, пока расстояния между вершинами укладываются в ограничение формулы (1). Теперь необходимо найти вторую такую группу и связать её с первой. Для этого «протягивается» дуга от одной из вершин первой группы до некоторой вершины, которая не входит ни в одну группу и ни в одну цепь. Такое «протягивание» называется «подъёмом к платам» по аналогии с расположением многоядерных процессоров на платах в суперкомпьютерах: уровень групп — это уровень многоядерных процессоров. На плате располагается несколько процессоров. Уровень плат — это совокупность верхних

уровней. Таким образом, при «протягивании» дуги осуществляется переход с нижнего уровня на верхний, чтобы через верхний уровень впоследствии добраться до следующего элемента нижнего уровня. Вес дуги выбирается минимальным из возможных, чтобы обеспечить «сборку» одной платы целиком перед тем, как начнётся «сборка» другой платы. Теперь со второго конца дуги надо добраться до очередной группы процессов-вершин на общей памяти. Так как не известно, на каком уровне иерархии находится конец дуги, приходится строить цепь из вершин и дуг (ещё не задействованных алгоритмом); веса дуг цепи также выбираются минимальными из возможных, причём вес каждой следующей дуги в цепи должен быть строго меньше веса предыдущей дуги. На очередном шаге станет невозможно найти такую дугу, и считается, что мы достигли вершины из следующей группы процессов-вершин на общей памяти, то есть добрались снова до нижнего уровня. Массив D нужен для того, чтобы ни одну вершину нельзя было просмотреть дважды.

Этап 2 не допускает циклов в локальной топологии, а потому плохо применим к архитектурам, в которых циклы есть (например, в которых процессоры соединены между собой непосредственно). Для восстановления циклов был введён этап 3: если суммарный вес дуг кратчайшего пути от вершины i до вершины j больше, чем Mij, и дуга i→j отсутствует, то данный цикл замыкается. На практике этап 3 состоит из n вызовов алгоритма Дейкстры нахождения кратчайшего пути из одной вершины во все остальные.

Особенности алгоритма: независимость от симметричности входной матрицы; выявление топологий произвольной структуры; результирующий граф имеет строго n вершин; результирующий граф является ориентированным; как будет показано далее, полиномиальная сложность от размеров матрицы.

Утв.1. Представленный алгоритм имеет в худшем случае временную сложность O(n3), где n — число процессов-вершин.

Доказательство утв.1 строится на том факте, что временная сложность этапа 2 не больше временной сложности этапа 3, которая равна n*O(n2) как следствие известной оценки для алгоритма Дейкстры.

Утв.2. Представленный алгоритм имеет в любом случае пространственную сложность O(n2).

Доказательство утв.2 сразу следует из размеров входной матрицы M, а также из максимальных размеров вспомогательных структур.

Смысл вершин графа, получаемого в результате работы вышеописанного алгоритма, полностью определяется смыслом строк и столбцов матрицы M. В данном случае строки и столбцы матрицы соответствуют MPI-процессам, запущенным на вычислительных узлах кластера. Отсюда пары дуг графа обозначают физические связи, которые могли бы соединять вычислительные узлы непосредственно. Коммуникационные элементы кластера — коммутаторы, маршрутизаторы — никак не учитываются. Существующие архитектуры могут как содержать в себе коммуникационные элементы, так и не содержать. В связи с этим граф объясняется двояко.

Будем называть *иерархической топологией* кластера топологию, в которой вычислительные узлы связаны между собой только посредством коммуникационных элементов. Напротив, *регулярной топологией* назовём топологию, в которой нет коммуникационных элементов, а вычислительные узлы соединены напрямую.

Таким образом, топология, выявляемая алгоритмом, является регулярной.

Получается, что если тест был проведён на кластере с регулярной топологией, алгоритм сможет выявить эту топологию. Если же тест проводился на кластере с иерархической топологией, алгоритм с большой вероятностью выдаст полносвязный граф. Более того, чем ближе полученный граф к полносвязному, тем лучше спроектирован кластер и тем удачнее реализация библиотеки MPI. Наличие дуг с неединичной вероятностью существования может говорить о потенциальных проблемах и/или недостатках кластера/библиотеки.

После решения задачи о выявлении графа встаёт задача о его отображении в трёхмерном пространстве для показа пользователю (то есть разработчику параллельных программ для кластеров).

К подсистеме визуализации графов предъявляется четыре требования: *сохранение относительной длины дуг графа, раскраска рёбер в зависимости от вероятности их существования, наглядность и скорость работы*. Первое требование не может быть выполнено в общем случае, так как всего 4 фиксированные точки при фиксированных длинах рёбер однозначно определяют в пространстве 5-ю точку, поэтому будет использован некоторый приближённый метод, удовлетворяющий данному требованию лишь частично.

Граф в трёхмерном пространстве изображается следующим образом: каждой вершине ставится в соответствие точка, а ребро представляется в виде отрезка, соединяющего две соответствующие вершины-точки. Таким образом, задача отображения графа сводится к расчёту трёхмерных координат точек. Данная формулировка соответствует классической задаче *многомерного шкалирования* [2] — отображение многомерных данных в пространство малой размерности. Для решения задачи вводится *функция стресса* — мера погрешности отображения расстояния между точками — и ищется её минимум. Координаты точек, соответствующие минимуму функции стресса, принимаются за ответ к задаче.

Предлагаемый алгоритм отображения графа в трёхмерном пространстве состоит из трёх этапов:

1) перевод матрицы M в матрицу длин рёбер (нормировка);

2) поиск координат вершин с помощью адаптированного метода многомерного шкалирования;

3) визуализация полученных точек и отрезков.

Этап 1. Пусть i и j — пара процессов-вершин, которым инцидентна дуга в выявленной топологии с ненулевой вероятностью существования. В матрице M этой паре соответствуют элементы Mij и Mji. Поскольку значения этих элементов, вообще говоря, различны, их нельзя использовать напрямую для вычисления расстояния между точками i и j (то есть для вычисления длины соединяющего их ребра). Также вычисление затрудняет тот факт, что исходный граф является ориентированным. Для получения длин рёбер и для визуализации матрицу M необходимо сделать симметричной, а граф — неориентированным. Сформируем ребро i–j как отождествление дуг i→j и j→i, а вес этого ребра положим равным

$$\frac{M_{ij} * p_{i\to j} + M_{ji} * p_{j\to i}}{p_{i\mapsto j} + p_{j\to i}} \quad (2),$$

где pk→l - вероятность существования дуги k→l. Вероятность существования ребра i–j положим равной сумме вероятностей существования дуг, отождествлённых с ним. Заменим значения Mij и Mji на результат формулы (2). Повторим те же действия для всех остальных пар вершин. При этом полагаем Mij:= Mji:=0, если дуг i→j и j→i нет. Выполнение вышеуказанных действий приведёт к тому, что матрица M станет симметричной, а граф — неориентированным. Далее осуществляется нормировка значений матрицы: ищется минимум среди всех значений, которым соответствуют рёбра с ненулевой вероятностью существования. Найденный минимум полагается равным 1 (то есть самое короткое ребро будет иметь длину 1), а все остальные значения в матрице нормируются по этому минимуму. Матрица M стала матрицей длин рёбер.

Этап 2. Потребуем, чтобы расстояния между вершинами, не соединёнными рёбрами с ненулевой вероятностью существования, были не меньше минимальной длины ребра (то есть 1). Данное требование делает классический метод многомерного шкалирования не применимым без некоторой адаптации.

Функцию стресса f будем представлять как (f1+f2)/2. f1 — часть из классического метода многомерного шкалирования, равная

$$f_1 = \sum_{\substack{i,j=0 \\ p_{i-j}>0}}^{n} ((x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2 - M_{ij}^2)^2 ,$$

где {xi,yi,zi} — трёхмерные координаты точки i, подлежащие определению, n — число точек, pi–j — вероятность существования ребра i–j. Таким образом, f1 описывает отклонение реального расстояния между точками i и j от желаемого расстояния Mij2.

Часть f2 нужна для удовлетворения поставленному требованию и представляется в виде

$$f_2 = \gamma * \sum_{\substack{i,j=0 \\ p_{i-j}=0,\ i\neq j}}^{n} f_{2_{ij}}$$ (обозначения те же, что и в f1), где

$$f_{2_{ij}} = \begin{cases} 1, & (x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2 = 0 \\ \dfrac{1}{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}, & (x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2 < 1 \\ 0, & (x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2 \geqslant 1 \end{cases}$$

Такая структура f2ij позволяет заострять внимание только на тех точках, которые надо удалить друг от друга. Параметр γ принимает значения из отрезка [0, 1] и обозначает уровень соблюдения требований.

Минимум функции f в текущий момент ищется с помощью градиентного спуска с переменным шагом. Заметим, что существуют целые трёхмерные подпространства исходного n-мерного пространства, на которых f принимает одинаковое значение: так как f зависит от координат лишь косвенно, их можно произвольным образом (одновременно) сдвигать и вращать. Начальные значения выбираются на прямой $x = y = z = t * n$, где t — параметр, затем придаются случайные смещения. Шаг спуска увеличивается в 16 раз, если на текущей итерации было найдено лучшее значение f. Иначе шаг уменьшается в 1.5 раза, а значения координат «откатываются» к полученным на предыдущей итерации. Числа 16 и 1.5 получены эмпирическим путём.

Этап 3. Отображаются только те рёбра i–j, для которых pi–j>0. Если канал, моделируемый ребром, дуплексный, то ребро изображается цилиндром малого радиуса. Если канал симплексный, то ребро изображается стрелкой, направление которой совпадает с направлением той отождествлённой с ребром дуги, которая имела большую вероятность существования. Цвет ребра i–j напрямую зависит от вероятности его существования и получается линейной интерполяцией между цветом фона и насыщенным зелёным цветом в RGB-представлении. Коэффициент интерполяции предложено рассчитывать по формуле

$$k = \frac{p_{i\mapsto j}^2 + p_{j\to i}^2}{p_{i\mapsto j} + p_{j\to i}}$$

Нетрудно проверить, что для k выполнено: $0 \leqslant min\{p_{i\to j}, p_{j\to i}\} \leqslant k \leqslant max\{p_{i\mapsto j}, p_{j\to i}\} \leqslant 1$.

Наконец, рассмотрим задачу сравнения выявленной топологии с эталонной топологией.

Так как топологии делятся на регулярные и иерархические (определения введены при описании задачи выявления), необходимо разработать два алгоритма сравнения для каждого вида.

Далее считается, что вершины графов топологий помечены непустой последовательностью символов — именем. Имя уникально только для вычислительных узлов, одинаковые имена определяют ядра одно и того

же процессора. Метка ребра задаёт пропускную способность физического канала, моделируемого данным ребром, в Гб/с.

Для представления эталонной топологии используется известный язык описания графов Graphviz/DOT [3].

Общий алгоритм сравнения топологий можно описать следующей последовательностью действий:

a) преобразование выявленной топологии: группировка и отождествление вершин с одинаковыми именами;
b) считывание dot-файла с эталонной топологией;
c) проверка того, что множество вершин выявленной топологии является подмножеством вершин эталонной топологии (сопоставление имён вершин);
d) подсчёт меры сходства топологий;
e) если мера сходства больше нуля, значит, эталонная топология имеет регулярную структуру; останов алгоритма; результатом является посчитанная мера сходства, выраженная в процентах;
f) мера сходства равна нулю, поэтому эталонная топология считается иерархической; формируется вспомогательный неориентированный граф по следующим принципу: если в выявленной топологии есть пара вершин i и j, соединённых дугой с ненулевой вероятностью существования, а в эталонной топологии существует ориентированный путь из i в j, то во вспомогательный граф добавляются вершины i, j и все вершины из соединяющего их кратчайшего пути в эталонной топологии; веса дуг формируются накоплением по определённой формуле (приведена ниже);
g) перевод разницы весов рёбер, принадлежащих одновременно графу эталонной топологии и вспомогательному графу, в цвета — раскраска графа эталонной топологии и его визуализация.

(a) При группировке вершин все дуги, соединяющие вершины внутри группы, уничтожаются, а все дуги, соединяющие вершины двух различных групп, отождествляются. Вес w получившейся дуги определяется в зависимости от текущих действий: если выполняется п. d, то вес равен максимуму из весов отождествляемых дуг; если выполняется п. f, то вес рассчитывается так: пусть G1 — первая группа вершин, G2 — вторая группа, тогда

$$w = \frac{\sum_{i \in G_1,\ j \in G_2} M_{ij}}{|G_1| * |G_2|} \quad (3),$$

где |G| - мощность множества G.

(c) Ошибка на данном этапе означает, что выявленная и эталонная топологии описывают совершенно разные коммуникационные среды либо разные подмножества одной среды.

(d) Мера сходства — количественная характеристика, позволяющая судить о совпадении двух топологий.

Введём обозначения: L — множество имён вершин, встречающихся и в выявленной топологии, и в эталонной топологии; E — множество рёбер в эталонной топологии, соединяющих вершины с именами из L; $\hat{V}$ — множество групп вершин в выявленной топологии, одна группа содержит вершины, именами которых является один элемент из L.

Тогда предлагаемая формула для расчёта меры сходства выглядит так:

$$sim = \frac{\sum_{G_1, G_2 \in \hat{V}} \left(\max_{i \in G_1,\ j \in G_2} p_{i \to j} \right)}{|E|} * 100\,\%$$

Объясним выбор именно такого вида формулы. Дело в том, что одному имени из L могут соответствовать сразу несколько вершин из выявленной топологии. Получаем, что каждому ребру из E могут соответствовать сразу несколько дуг из выявленной топологии. Для проведения сравнения необходимо отождествить все такие дуги. Вероятность существования результирующей дуги полагается равной максимуму из вероятностей существования отождествляемых дуг. Результирующая дуга взаимно однозначно соответствует какому-то элементу из $\hat{V}$, а каждый элемент из $\hat{V}$ — элементу из E; назовём вероятность существования этой дуги вероятностью существования элемента из $\hat{V}$. Таким образом, элемент из $\hat{V}$ похож на ребро из E на величину вероятности существования элемента из $\hat{V}$, ведь все рёбра из E имеют единичную вероятность существования. Итак, мера сходства двух топологий складывается из порёберной схожести и равна среднему арифметическому вероятностей существования элементов из $\hat{V}$, выраженному в процентах.

(f) Рассмотрим тактику накопления весов рёбер во вспомогательном графе. Пусть i и j — пара вершин во вспомогательном графе, добавленных в него по принципу формирования (описан в п. f), тогда r — кратчайший ориентированный путь из i в j, причём и дуги, и вершины этого пути принадлежат как вспомогательному графу, так и эталонной топологии. Обозначим через rk k-ю дугу пути (считая по порядку от вершины i), |r| - число дуг в пути. w — величина, рассчитанная по формуле (3), где G1 формируется из одноимённых вершин для вершины i, G2 — для вершины j.

Каждое ребро в эталонной топологии помечено пропускной способностью соответствующего канала. Величина w играет роль времени прохода сообщения по пути r (считается, что все сообщения i от j до идут по

кратчайшему пути). Идея состоит в том, чтобы распределить w каждой дуге из r обратно пропорционально её пропускной способности. То есть величина ak, которую надо прибавить к весу дуги rk, будет равна

$$a_k = w * \frac{\frac{1}{w_{r_k}}}{\sum_{i=0}^{|r|-1} \frac{1}{w_{r_i}}} \quad (4),$$

где wrk — вес дуги в эталонной топологии, соответствующей rk.

Итак, каждая дуга во вспомогательном графе накапливает вес по формуле (4), а также запоминает количество путей r, которые содержат данную дугу. После прохождения по всей эталонной топологии и, соответственно, по всей выявленной топологии, веса дуг i→j и j→i складываются и делятся на суммарное запомненное количество путей, тем самым вес ребра i–j (так как вспомогательный граф, на самом деле, неориентированный) становится равным среднему времени прохождения сообщений через данное ребро.

(g) Получаем, что веса рёбер во вспомогательном графе задают «реальную» пересылку сообщений (т. к. значение w складывается из значений исходной матрицы расстояний), а веса рёбер в эталонной топологии — «идеальную» пересылку. Однако, «идеальная» пересылка задана пока что пропускными способностями. Чтобы привести веса в обоих графах к одному формату, необходимо ввести некое сообщение определённой длины, которое будет «передаваться» по рёбрам эталонной топологии. Длину этого сообщения можно определить, исходя из матрицы M (ведь матрица M была получена для какой-то длины сообщений либо для комбинации длин). Пересчитаем веса рёбер эталонной топологии как отношение длины сообщения к пропускной способности. Теперь веса рёбер в обоих графах обозначают время передачи сообщений.

На основании полученных данных при визуализации рёбрам эталонной топологии назначаются цвета. Имеют место 3 случая: 1) ребро входит в эталонную топологию, однако не содержится во вспомогательном графе — назначается простой цвет (белый); 2) вес ребра во вспомогательном графе не больше веса такого же ребра в эталонной топологии — назначается красный цвет; 3) вес ребра во вспомогательном графе строго больше веса такого же ребра в эталонной топологии — назначается синий цвет. В зависимости от абсолютного значения разности весов цвет будет иметь разную насыщенность.

Чтобы выполнить визуализацию эталонной топологии, необходимо запустить для неё алгоритм отображения графов, посчитав новую матрицу M: если ребро i–j принадлежит эталонной топологии, то Mij:=Mji:=<вес ребра i–j>, иначе Mij:=Mji:=0.

Все вышеперечисленные алгоритмы реализованы в составе программы Network Viewer 2 [4], входящей в комплекс программ PARUS. Это объясняется тем, что в Network Viewer 2 уже присутствуют средства для визуализации результатов тестирования кластеров. Реализация выполнена на языках C, C++, пользовательский интерфейс написан с помощью библиотеки Qt, визуализация графов выполняется с помощью библиотеки OpenGL, предоставляемой Qt-модулем QtOpenGL. Распараллеливание кода производится с использованием технологии OpenMP.

В таблицу 1 сведено среднее время работы алгоритма выявления топологии для различных архитектур на машине с процессором AMD Phenom II (1 ядро).

Таблица 1. Среднее время работы алгоритма выявления топологии для одной матрицы

Кластер	Число узлов	Среднее время для одной матрицы
BlueGene/P	128	10 мс
	256	80 мс
	1024	760 мс
«Ломоносов»	500	90 мс
«Чебышёв»	64	5 мс
	128	9 мс
MVS-50k	300	40 мс
switch	38	0.04 мс

На примере BlueGene/P хорошо просматривается кубическая зависимость времени работы алгоритма от числа процессов.

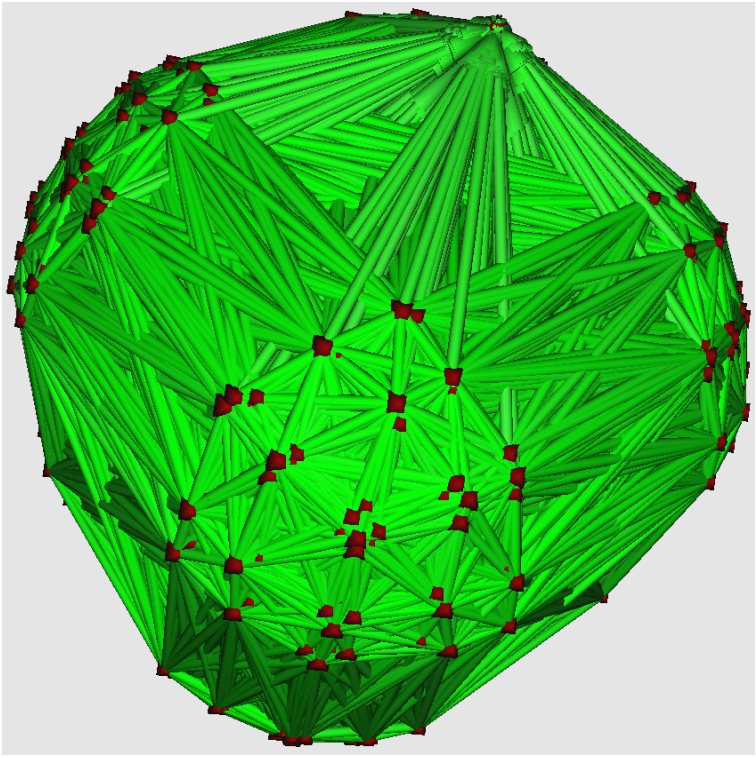

Рис. 1. "BlueGene/P", 512 процессов, матрица для длины сообщений 0

На рис. 1 отображены связи между 512 процессорами суперкомпьютера BlueGene/P. Для вычисления длин рёбер использовалась матрица, соответствующая длине сообщений 0. BlueGene/P имеет регулярную структуру, то есть процессоры соединены каналами связи непосредственно друг с другом. Network Viewer 2 смог показать это. Обратим внимание на одну вершину, отличающуюся от остальных: она не вписывается в общую структуру, и к ней ведут стрелки, а не отрезки, что означает слишком долгую посылку сообщений от данной вершины. Оказывается, этой вершине соответствует MPI-процесс номер 0. Известно, что MPI выделяет 0-й процесс среди всех процессов, поэтому возможно ухудшение его коммуникационных характеристик.

Рис. 2 изображает связи между ядрами процессоров в суперкомпьютере «Ломоносов». При тестировании один MPI-процесс запускался на одном ядре каждого 8-ядерного процессора. Для расчёта длин рёбер были взяты средние арифметические расстояний из всех входных матриц. Нижний порог вероятности существования рёбер установлен в 30%.

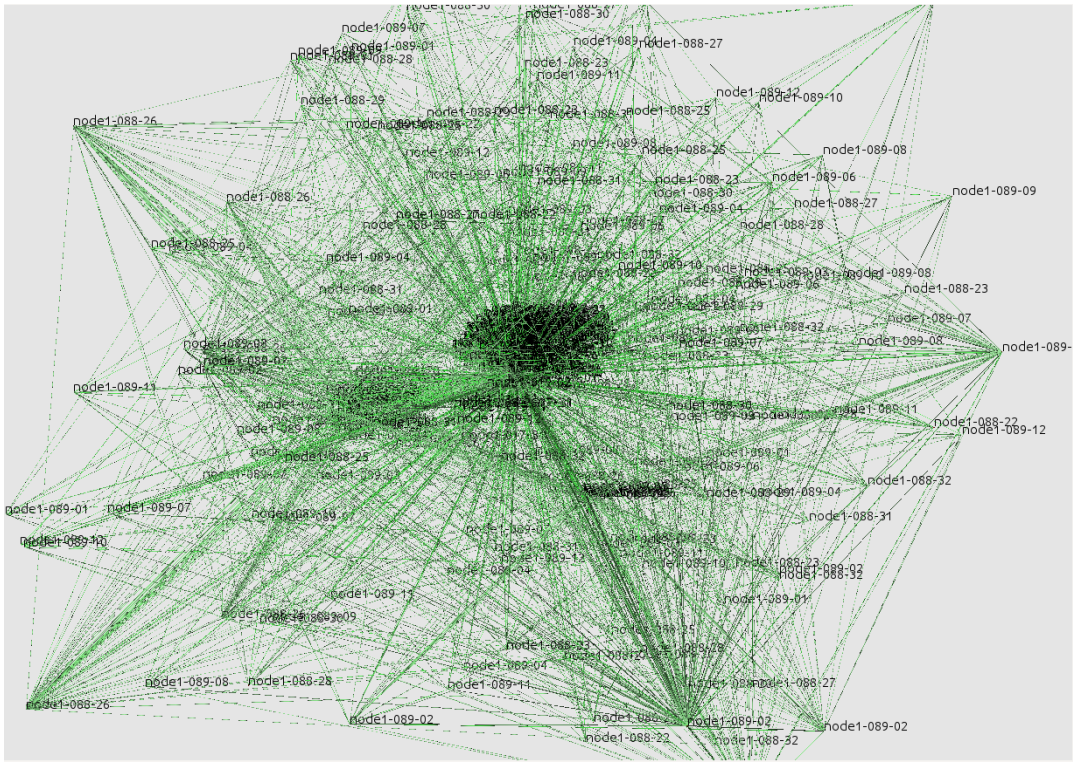

Рис. 2. "Ломоносов", 500 процессов, ср. арифм., вероятность существования дуг >= 30%

Можно видеть сгущение вершин у визуального центра графа. Это объясняется особенностью топологии суперкомпьютера «Ломоносов». Она организована иерархически, но таким образом, чтобы время передачи сообщений между любыми двумя вершинами было примерно одинаково. Алгоритм выявления топологии в данном случае выдаёт полносвязный граф, как раз сосредоточенный в центре рисунка. Расположение остальных вершин можно отнести к погрешностям в длинах рёбер и к неточности работы алгоритма отображения.

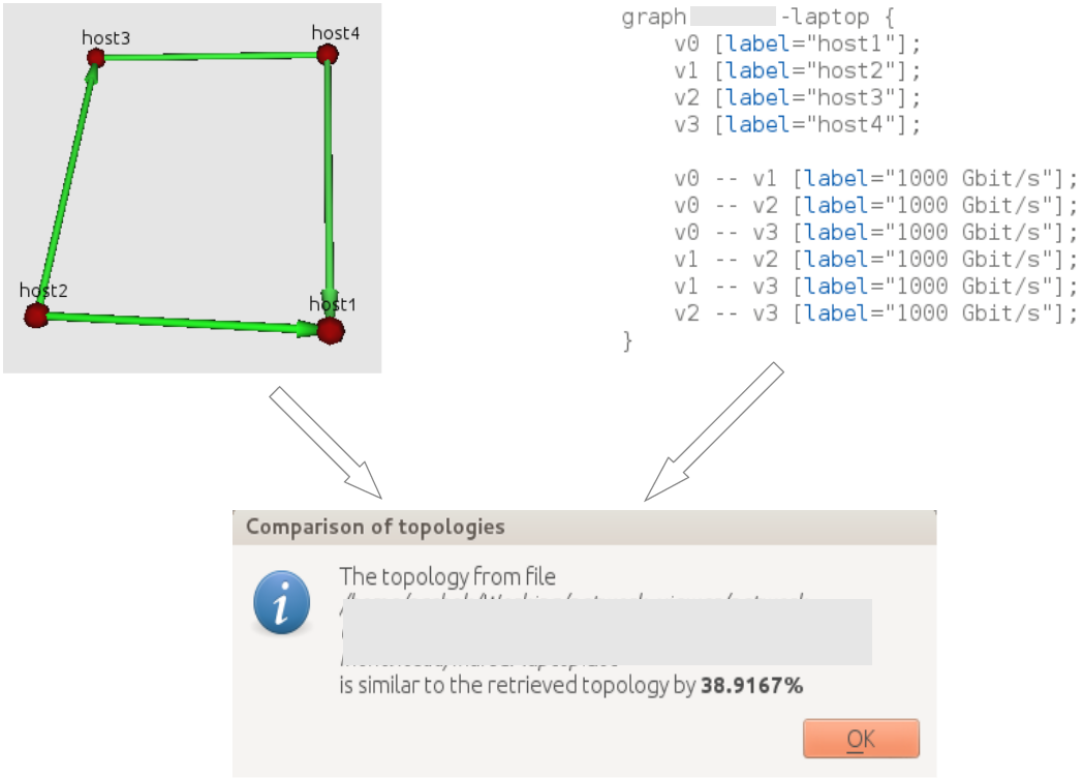

Рис. 3. Результат сравнения выявленной и эталонной топологий

На рис. 3 слева вверху показана некая выявленная топология, справа вверху — формат задания эталонной топологии для той же самой архитектуры, а внизу — результат сравнения этих двух топологий.

Здесь эталонная топология задаётся как полносвязный граф. Так как топологии регулярны, выводится сообщение со значением меры сходства.

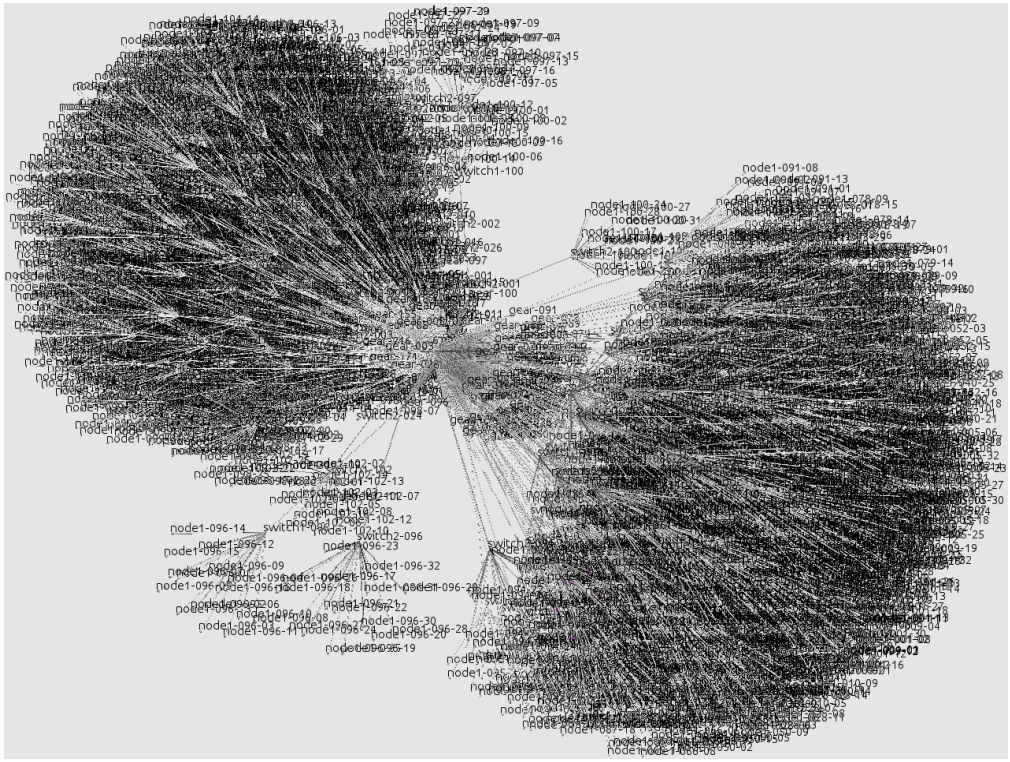

Рис. 4.Эталонная топология суперкомпьютера "Ломоносов"

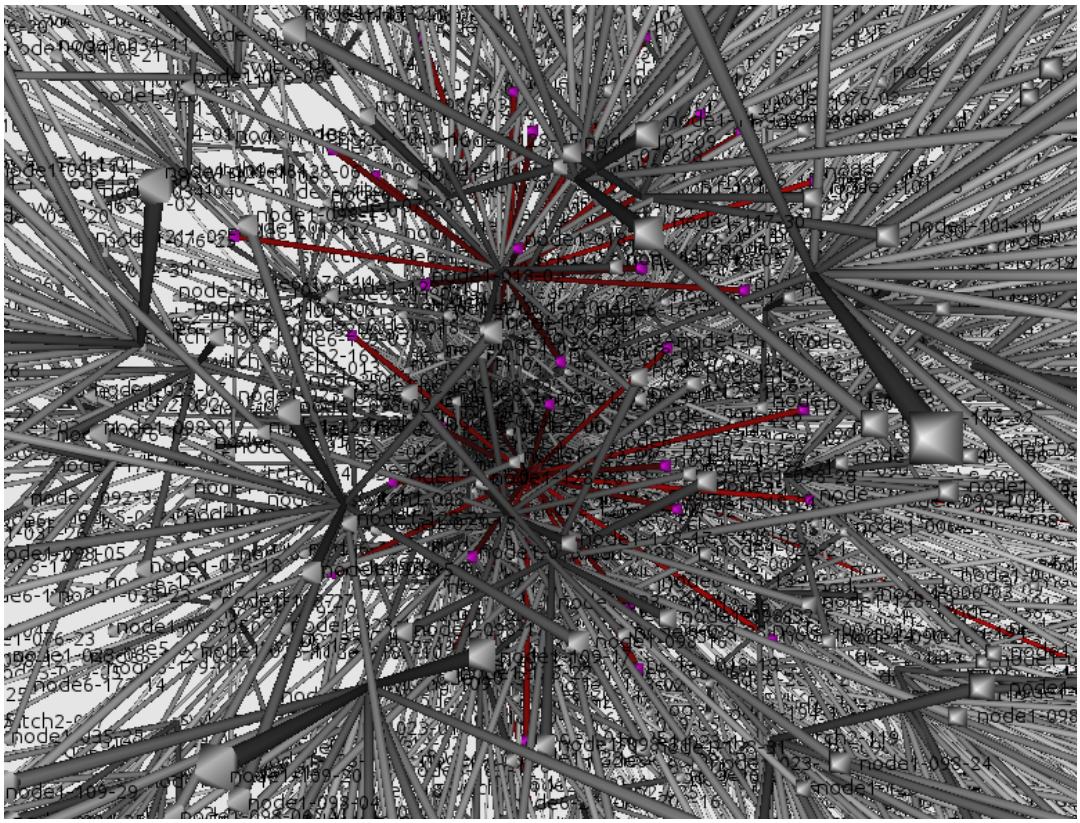

Рис. 5. Результат сравнения выявленной топологии "Ломоносова" с эталоном

На рис. 4 показан построенный в программе Network Viewer 2 граф коммуникационной среды суперкомпьютера «Ломоносов». Топология взята из спецификации. А на рис. 5 показана часть различий между выявленной топологией с рис. 2 и эталонной топологией с рис. 4. Ребра имеют насыщенный красный цвет, что говорит от большом отклонении «реальной» пропускной способности тех каналов от «идеальной». В качестве

«идеальной» пропускной способности бралась пиковая пропускная способность сетей InfiniBand, используемых в «Ломоносове», равная 32 Гб/сек (режим QDR). Чтобы определить трёхмерные координаты вершин графа на рис. 5, потребовалось около 15 часов на машине с процессором AMD Phenom II (задействовано 1 ядро).

Исходный код Network Viewer 2 доступен через git-репозиторий по адресу http://git.code.sf.net/p/parus/code, далее папка /network_test2/src/network_viewer_qt_v2/.

Работа частично поддержана грантом РФФИ 12-07-31088.

ЛИТЕРАТУРА:

1. А.Н. Сальников, Д.Ю. Андреев, Р.Д. Лебедев "Инструментальная система для анализа характеристик коммуникационной среды вычислительного кластера на основе функций стандарта MPI" // Вестник Московского университета, № 1, Москва, изд-во МГУ, 2012 г., с.39–48
2. I. Borg, Patrick J.F. Groenen "Modern Multidimensional Scaling — Theory and Applications" // Springer Series in Statistics, Springer Science+Business Media, Inc, 2005, p. 261
3. The DOT language // [электронный ресурс; 28.05.2013]: http://www.graphviz.org/content/dot-language
4. А.Н. Сальников, П.С. Банников "Система визуализации результатов MPI-тестирования коммуникационной среды вычислительных комплексов" // Вестник Пермского университета. Математика. Механика. Информатика, № 3, 2012 г., изд-во ПГНИУ, с.80-85