УДК 004.75

В.В. Воронина, С.О. Смеречинский

СИСТЕМА МОДЕЛИРОВАНИЯ КЛАСТЕРА, ИМИТИРУЮЩАЯ ПРОЦЕССЫ ЗАДАЧ АНАЛИЗА BIGDATA

Воронина Валерия Вадимовна, *кандидат технических наук, окончила факультет информационных систем и технологий Ульяновского государственного технического университета. Доцент кафедры «Информационные системы» УлГТУ. Имеет статьи в области интеллектуального анализа временных рядов и разработки систем поддержки принятия решений. [e-mail: vvsh85@mail.ru].*

Смеречинский Сергей Орестович, *магистр, окончил магистратуру факультета информационных систем и технологий УлГТУ. Инженер-программист ООО «АИС Город». Имеет статьи в области разработки программного обеспечения. [e-mail: quigon173@gmail.com].*

Аннотация

Данная статья рассматривает предметную область обработки больших объемов данных с помощью кластерных систем. В работе указываются существующие способы обработки BigData, на которых основывается предлагаемое решение повышения эффективности работы кластера. В качестве базовых решений, на которых строятся и предлагаются способы повышения эффективности работы кластера, были взяты технология Hadoop и методология MapReduce. Эффективность работы кластерных систем предполагает рассмотрение процессов работы кластера в виде иерархической архитектуры, состоящей из трех уровней: уровня узла кластера, уровня сегмента кластера и уровня топологии кластера. Статья рассматривает возможные способы повышения эффективности распределения вычислительных ресурсов кластера за счет комбинации вариантов выбора топологии построения кластера, использования более эффективного алгоритма распределения нагрузки и применения графических процессоров, которое подразумевает распределение вычислительных нагрузок между CPU и GPU. Предлагаемые в работе рекомендации и выводы подтверждены экспериментально.

Ключевые слова: Big Data, кластер, алгоритм балансировки, топология, CPU, GPU, MapReduce.

THE SYSTEM FOR MODELLING THE CLUSTER SIMULATING PROCESSES OF BIG DATA ANALYSIS

Valeriia Vadimovna Voronina, *Candidate of Engineering; graduated from the Faculty of Information Systems and Technologies of Ulyanovsk State Technical University; Associate Professor at the Department of Information Systems of Ulyanovsk State Technical University; an author of articles in the field of intellectual analysis of time series. e-mail: vvsh85@mail.ru.*

Sergei Orestovich Smerechinskii, *Magister; finished his Master's studies at the Faculty of Information Systems and Technologies of Ulyanovsk State Technical University; Software Engineer at "AIS Gorod" Company; an author of articles in the field of software development. e-mail: quigon173@gmail.com.*

Abstract

The subject area for processing Big Data with the use of cluster systems is described in the article. The paper examines the existing methods of processing Big Data on which the proposed solution for increasing the efficiency of the cluster functioning is based. As basic solutions for building and proposing ways to increase the efficiency of the cluster, the Hadoop technology and the MapReduce methodology were chosen. The efficiency of cluster systems involves consideration of cluster processes in the form of the hierarchical architecture consisting of three levels: cluster node level, cluster segment level and cluster topology level. The paper indicates solutions that can be useful in order to improve the efficiency of cluster resource allocation by choosing the topology of cluster construction, using the more efficient load-balancing algorithm, and using graphics processors, which involves distributing computational loads between CPU and GPU. Recommendations proposed in the article are verified experimentally.

Key words: Big Data, cluster, balancing algorithm, topology, CPU, GPU, MapReduce.

Введение

В настоящее время задачи эффективной обработки больших объемов данных являются как никогда актуальными. Различные способы их решений сформировали область знаний под названием BigData. BigData – совокупность подходов, инструментов и методов обработки структурированных и неструктурированных данных огромных объёмов и значительного многообразия для получения воспринимаемых человеком результатов, эффективных в условиях непрерывного прироста, распределения по многочисленным узлам вычислительной сети [1]. BigData подразумевает обработку колоссальных объемов данных, которые невозможно разместить на одной машине и произвести достаточно быстро их обработку одним процессором. В итоге данные и операции по их обработке чаще всего распределяются на несколько машин. Объединение нескольких машин в группу, которая выполняет узкоспециализированную задачу, называется кластером. Составляющие кластера соединяются высокоскоростным каналом связи, позволяющим быстро передавать данные, а для пользователя кластер выступает как единый аппаратный ресурс [2]. Причем чаще всего при работе с BigData для организации обработки данных используется методология MapReduce [3]. Таким образом, если рассматривать вопрос обработки данных с точки зрения «железа», то для повышения его эффективности необходимо построить эффективно работающий кластер. Одним из способов решения данной задачи является реализация имитационной модели кластера, которая может быть использована в вычислительных экспериментах для определения значений характеристик, влияющих на эффективность обработки данных.

Кроме того, при работе кластеров важным является использование эффективного алгоритма распределения нагрузки. Большое распространение получил алгоритм по WeightRoundRobin [4], который подразумевает распределение задач между узлами, основываясь на числовых коэффициентах каждого узла.

Проблема эффективности построения кластера рассматривалась в работах [5–10]. По результатам их анализа были выявлены следующие определяющие факторы:

а) топология кластера;

б) распределение нагрузки между узлами;

в) способы ускорения работы узлов.

Однако в указанных работах обозначенные выше факторы рассматривались чаще всего без взаимосвязи. То есть повышение эффективности велось лишь в каком-то одном направлении. В настоящей же работе рассмотрено объединение факторов и построение на их основе трехуровневой модели кластера, позволяющей определять способы повышения эффективности обработки данных на всех трех уровнях.

В описании модели будут использоваться следующие термины:

• Центральный узел – узел кластера, который принимает все внешние запросы от пользователей системы, передает информацию по запросам управляющим узлам и возвращает результаты выполнения запросов пользователям;

• Управляющий узел – узел кластера, который осуществляет управление работой исполняющих узлов при выполнении запросов, следит за уровнем распределения нагрузки и выравнивает ее;

• Исполняющий узел – узел, осуществляющий выполнение всех поставленных задач, используя те данные, которые хранятся на нем или были получены в результате распределения.

Рассмотрим подробнее трехуровневую модель кластера и приведем данные вычислительных экспериментов, доказывающих ее состоятельность.

Основные элементы модели кластера

Для формального определения модели кластера необходимо явно выявить ее элементы. Она определяется трехуровневой структурой, которая подразумевает переход от общего к частному. Таким образом, модель кластера подразумевает следующую иерархию уровней:

1. Топология кластера;
2. Роли узлов кластера и каналы передачи данных;
3. Вычислительные особенности узлов.

Рассмотрим их подробнее, начав с верхнего уровня.

Топологию любого кластера формально можно представить в виде графа (обозначим его DC) следующим образом:

$$DC:=(N,C),\ n \to c,\ c \in C,\ n \in N, \quad (1)$$

где DC – кластер, N – множество узлов кластера, C – множество каналов передачи данных, $n \to c$ – отношение, определяющее, что к каждому узлу подключается кабель для передачи данных так, чтобы каждый узел мог получить для работы любые данные с других узлов.

Определим для топологии кластера критерий эффективности (оптимальности) построения следующим образом. Пусть оптимальная топология строится так, чтобы общее количество узлов было минимальным, количество загруженных узлов равнялось нулю и при этом сохранялся принцип локальности данных.

Под принципом локальности данных подразумевается, что множество данных, присланных на обработку узлу N_x кластера, должно обрабатываться либо на самом узле N_x, либо на узле, который находится максимально близко к узлу N_x.

Тогда формально характеристики оптимальной топологии можно записать так:

$$\begin{aligned} &\forall\ x \to n_x \mid x \in X \quad n_x \in N \quad \exists\ n_i \in N, \\ &DC_{xi} = (N_{xi},\ C_{xi}),\ |DC_{xi}| \to \min,\ N_{xi} \subset N, \\ &|N_{ol}| = 0,\ N_{ol} \subset N_{xi}, \end{aligned} \quad (2)$$

где X – множество данных; N – множество узлов; n_x – узел, хранящий данные; n_i – узел, который будет данные обрабатывать; DC_{xi} – подграф, в рамках которого

будут передаваться данные, N_{ol} – множество перегруженных узлов.

То есть для любого объема данных данные x, хранящиеся на узле n_x, где x принадлежит множеству данных X, а n_x – множеству узлов N, может быть определен узел n_i для обработки данных x так, чтобы количество узлов в множестве узлов N_{xi}, через которые будут передаваться данные, было минимальным. Причем граф N_{xi} должен строиться таким образом, чтобы в перечне узлов, участвующих в передаче и обработке данных, не было перегруженных.

Указанные ограничения будут использоваться в имитационной модели для определения количества узлов топологии и соединения их каналами передачи данных.

Теперь перейдем к роли **узлов кластера и каналов передачи данных**. Все узлы кластера обладают общими характеристиками, такими как скорость обработки данных и объем оперативной памяти, однако в зависимости от своей роли в кластере они могут обладать дополнительными ключевыми свойствами, такими как количество принимаемых внешних запросов в секунду, алгоритм распределения, скорость связи с управляющими узлами, скорость инициализации задач на обработку данных, скорость управления потоками данных, скорость управления порядком выполнения задач и объем их очереди, скорость формирования результата выполнения задачи, скорость чтения данных с носителя, скорость обработки данных графическим процессором, скорости выполнения Map-задач, Reduce-задач и стадии FinalReduce.

Если переходить к формальному представлению описанного, то можно дополнить представленную выше модель. Как было определено ранее, кластер состоит из узлов N, где каждый элемент множества принадлежит какому-то из подмножеств N_c, N_u, N_w. Здесь N_c – множество центральных узлов, которое обычно состоит из одного элемента, N_u – множество управляющих узлов, N_w – множество исполняющих узлов. Каждый узел имеет ряд характеристик, влияющих на его работу (объем оперативной памяти, количество процессоров и т. д.). Эти характеристики могут быть описаны множествами p_c, p_u, p_w соответственно. Причем существует такое множество базовых характеристик P_b (характерных для всех узлов), что $p_b = p_c \cap p_u \cap p_w$.

Тогда узлы можно представить с помощью их характеристик следующим образом:

$$N_c = \begin{bmatrix} p_{c1} \\ \vdots \\ p_{ck} \\ \vdots \\ p_{cn} \end{bmatrix}, \; N_m = \begin{bmatrix} p_{u1} \\ \vdots \\ p_{uk} \\ \vdots \\ p_{um} \end{bmatrix}, \; N_w = \begin{bmatrix} p_{w1} \\ \vdots \\ p_{wk} \\ \vdots \\ p_{wl} \end{bmatrix}, \quad N_c \cup N_u \cup N_w = N, \tag{3}$$

где k – мощность множества p_b; n, m, l – мощность множеств N_c, N_u, N_w соответственно: $p_c \in P_c$, $p_u \in P_u$, $p_w \in P_w$.

Данные между узлами передаются через каналы связи, обладающие следующими характеристиками:

- пропускная способность;
- тип протокола передачи данных.

То есть описанную модель необходимо дополнить множеством каналов передачи данных C, соединяющих элементы кластера. Каналы передачи данных можно охарактеризовать с помощью множества свойств P_c следующим образом:

$$C = \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix}, \tag{4}$$

где n – мощность множества P_c, $c \subset P_c$.

Следовательно, способом повышения эффективности работы кластера на данном уровне будет поиск алгоритма оптимального распределения задач по узлам, исходя из их характеристик, характеристик каналов связи и с учетом определенных ограничений топологии.

Здесь стоит отметить такие элементы построения сети, как шлюзы и коммуникаторы. Данные устройства физически не представляют собой среду передачи данных, однако они также обладают своей пропускной способностью. Таким образом, в рамках нашей модели коммуникаторы можно рассматривать как каналы передачи данных.

Третьим уровнем нашей модели являются **вычислительные особенности узлов**, а именно множества CPU и GPU (центральных и графических процессоров), где каждый процессор обладает своими характеристиками, как-то скорость обработки данных, зависящей от объема и характера данных. Следовательно, на данном уровне эффективность может быть повышена дополнением алгоритма распределения данных между GPU и CPU. То есть при поступлении запроса на обработку существует перечень данных, которые можно распределить для обработки между элементами множества процессоров так, чтобы время выполнения было минимальным. Формально данное распределение с критерием его эффективности можно записать следующим образом:

$$\forall c \in PU_c, \; g \in PU_g,$$
$$PU = PU_c \cup PU_g, f : PU \xrightarrow{x} t,$$
$$\exists X = \left\{X_{c1},\ldots,X_{cn},X_{g1},\ldots,X_{gm}\right\}$$
$$\left|\max\left(f(X_{c1}),\ldots,f(X_{cn}),f\left(X_{g1}\right),\ldots,f\left(X_{gm}\right)\right)\right| \to \min, \tag{5}$$

где PU_c и PU_g соответственно множества центральных и графических процессоров и $PU = PU_c \cup PU_g$; n и m – мощности множеств PU_c и PU_g; X_i – перечень за-

дач для обработки; f – это отношение, сопоставляющее время обработки объема данных на процессоре.

То есть для любого c и g, которые принадлежат множеству центральных процессоров PU_c и графических процессоров PU_g соответственно, где каждый процессор имеет свою скорость обработки данных f, существует такое распределение задач между процессорами X (где X_{ci} – набор данных для обработки на центральном процессоре i, X_{gj} – набор данных для обработки на графическом процессоре j), что время обработки перечня X должно стремиться к минимуму, причем временем выполнения считается наибольшее время обработки данных X_m на процессоре PU_m.

Алгоритмы имитационной модели

В рамках созданной модели были выделены следующие операции кластера:

1. Внешние запросы: запрос на выполнение задачи и запрос на изменение данных;
2. Внутренние запросы: инициализация задачи, взаимодействие между управляющими узлами, взаимодействие между управляющим и выполняющими узлами, задача распределения данных;
3. Перемещение данных: перенос данных между узлами, перенос данных в виде пар ключ-значение, сгенерированных на стадии Map, передача промежуточных данных в виде пары ключ-значение.

Каждая операция подразумевает передачу данных x, причем различного размера $S = f(x)$. Все внутренние запросы имеют фиксированный размер $S_s << S$, где S – размер данных, хранящихся в кластере.

Для других процессов размер данных является непостоянной величиной, не зависящей от других процессов. Можно считать, что значение S_i имеет случайный характер. Размер данных операций находится в границе $S_{min} \le S(x) < S_{max}$. Таким образом, для получения значения размера правильнее использовать генератор случайных чисел в диапазоне $[S_{min}; S_{max}]$.

При получении запроса на выполнение задачи происходит следующая последовательность операций:

1. Передача задачи узлам (размер передаваемых данных – случайное значение в диапазоне $[S_{min1}; S_{max1}]$);
2. Взаимодействие между управляющими узлами для назначения задачи (размер данных S_2 для каждого из узлов);
3. Передача инициализированной задачи (размер данных определяется выражением $S_2 + S_3$);
4. Чтение данных и запуск Map-стадии (допустимый диапазон значений: $[0; Z * S_4]$, где S_4 – размер фрагмента данных, Z – количество фрагментов для переноса);
5. Распределение по исполняющим узлам Key-Value значений для нормирования (допустимый диапазон значений: $[0; M * S_5]$, где S_5 – размер пары ключ-значение, M – количество пар);
6. Запуск Reduce и сбор результатов на одной машине ($K * S_6$, где K – количество узлов, S_6 – размер данных, который получился после выполнения стадии Reduce на каждой машине);
7. Возвращение результата (размер данных фиксирован величиной S_7 – размером данных, полученных после «суммирования»).

При выполнении операций учитывается допустимая пропускная способность каналов связи C, а также размер оперативной памяти, в которую сохраняются данные после передачи в узлах N. Зная пропускную способность каналов передачи данных и размер памяти узлов, будет возможно определить элементы, перегруженные данными.

Для определения эффективного распределения запросов по узлам кластера нужно решить следующую задачу. Пусть существует множество задач L, которые необходимо распределить по N узлам кластера. Формально это можно определить следующим образом:

$$\begin{aligned} &\forall n \in N, l_b \in L_b \mid L_b \to N, c : n \xrightarrow{l} t, \\ &\exists L = \{l_1, l_2, \ldots, l_k\}, \; L \cap L_b = \varnothing, \\ &k = |N|, \; T = \min C(L_b + L), \end{aligned} \tag{6}$$

где N – множество узлов, L_b – множество текущего уровня нагрузки узлов, c – функция времени выполнения перечня задач l на узле n.

То есть для любого узла кластера n с уровнем текущей нагрузки l_b, c – функцией времени выполнения перечня задач l, существует такое распределение задач L между узлами кластера, что время выполнения задач после распределения с учетом начального уровня нагрузки будет минимальным. Для определения эффективного распределения требуется найти T. В представленной модели данная задача решается алгоритмом, который позволяет получить эффективное распределение задач между исполняющими узлами кластера. В качестве входных данных алгоритм принимает перечень задач для распределения и данные о текущей нагрузке на каждом процессоре. Выходными данными является множество групп задач, распределенных между узлами.

Алгоритм:

1. Определить исходное выражение для $L_b = \{L_{b1}, L_{b2}, \ldots, L_{bk}\}$. При запуске системы моделирования инициализировать $L_{b1} = L_{b2} = \ldots = L_{bk} = 0$;
2. $Ln_{i+1} = Ln_i + 1$;
3. Определить $min(max(f_1(L_{1(i+1)}), f_2(L_{2i}), \ldots, f_N(L_{ki})), \ldots, max(f_1(L_{1i}), f_2(L_{2i}), \ldots, f_N(L_{k(i+1)})))$, где $f(L_i)$ – время выполнения перечня задач L_i, притом L содержит набор данных.
4. Запомнить индекс узла, на котором перечень задач выполняется быстрее;

5. Выполнять, пока задачи из множества L не будут распределены.

Здесь L_b – базовое распределение задач, L_{n_i} – перечень распределенных задач на i-й итерации.

Для определения эффективности распределения данных между GPU и CPU необходимо решить следующую задачу: пусть существует множество задач L, которые необходимо обработать на узле кластера. В узле установлены CPU и GPU, где время обработки задач определяется функциями $c(L_c)$ и $g(L_g)$ соответственно. $s(L)$ – функция времени выполнения задач на узле. Тогда

$$s(L) = min(c(L_c), g(L_g)), \quad (7)$$

где L_c и L_g соответственно набор задач для обработки CPU и GPU, $L = L_c + L_g$.

Для определения эффективного распределения требуется найти $min(s(L))$. Данную задачу можно решить следующим образом:

1. Определить исходное выражение:
   $L = \{L_{c0},\ L_{g0}\}$;
2. $L_{i+1} = L_i + 1$;
3. Определить
   $min(max(c(L_{ci+1}), g(L_{gi})),\ max(c(L_{ci}), g(L_{gi+1})))$;
4. Запомнить индекс процессора, на котором перечень задач выполняется быстрее;
5. Выполнять, пока задачи из множества L не будут распределены.

Реализация системы моделирования

На основе представленной модели кластера была разработана система моделирования, которая позволяет:

- Смоделировать работу кластера DC из узлов N, соединенных между собой каналами C, с учетом характеристик узлов P;
- Обнаружить уязвимые и нагруженные элементы кластера n_q и/или c_q;
- Выбрать топологию кластера для его эффективной работы;
- Сравнить эффективность алгоритмов распределения ресурсов при выполнении задач.

За основу были взяты элементы фреймворка Hadoop и взаимодействия узлов кластера и процессов в них. Подразумевалось, что кластер DC имеет топологию построения сети, по которой связываются узлы N. Однако, в Hadoop не делается акцент на топологию, а структура кластера представляется, как показано на рисунке 1 [2]. где $к$ – коммуникатор, $c1,\ c2$ – сегменты, $y1,\ y2,\ y3$ – узлы, d – диски. Важно заметить, что узлы объединяются в некоторые группы, называемые сегментами.

Разработанное программное обеспечение подразумевает, что топология кластера может быть трех разных типов, в зависимости от поставленных задач L и количества узлов. Выделяются топологии: «Звезда» (рис. 2), «Двухуровневое дерево» (рис. 3), «Трехуровневое дерево» (рис. 4). На рисунках используются обозначения: c – центральный узел, u – управляющий узел, I – исполняющий узел. Линиями обозначаются каналы передачи данных, а места пересечения линий обозначают соединение коммуникаторами.

Программное обеспечение реализовано на языке программирования высокого уровня C++ с использованием кроссплатформенного фреймворка Qt в виде десктопного приложения. Программное обеспечение собрано под платформу Windows x64. Кроме того, для работы с графическими процессорами использовался фрейворк OpenCL.

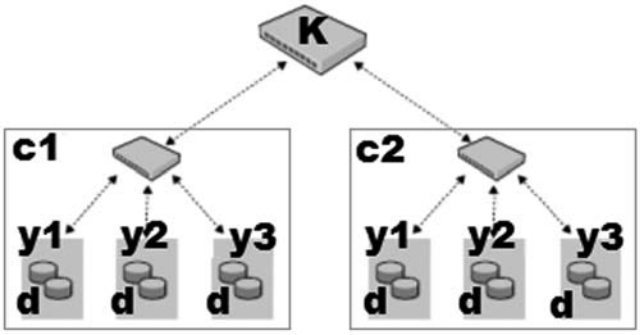

Рис. 1. Структура кластера DC в Hadoop

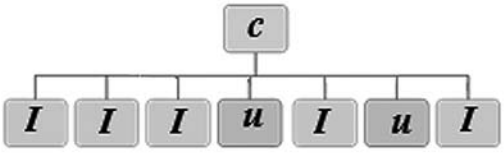

Рис. 2. Кластер DC с топологией «Звезда»

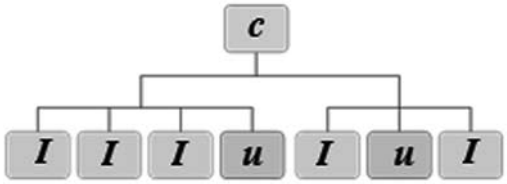

Рис. 3. Кластер DC с топологией «Двухуровневое дерево»

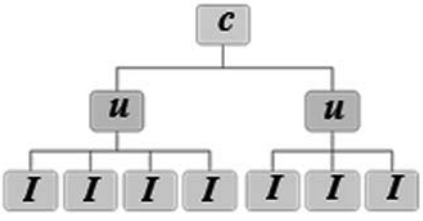

Рис. 4. Кластер DC с топологией «Трехуровневое дерево»

Для организации работы системы в рамках модели было введено понятие задачи l. Задача подразумевает выполнение действия в течение времени t, которое может осуществляться либо в процессе выполнения запроса пользователя кластеру, либо вызова системных операций, либо производится распределение задач и т. д. Для имитации работы кластера введено абстрактное понятие тик, которое подразумевает выполнение какой-то атомарной операции. Каждая задача имеет измерение в тиках, где время тика t_a, то есть подразумевается, что выполнение задачи сложностью V осуществляется за время V^*_{ta}. Каждый узел n выполняет определенное количество операций в секунду, соответственно, если рассматривать в рамках выполнения атомарных операций, то в секунду компьютер может осуществить M тиков. Помимо задач и тиков реализованы абстракции каналов связи c, узлов n и оборудования сети и другие. Всего в системе реализовано 5 классов абстракций.

Реализованы следующие абстракции задач. Задача Map генерируется на основе внешнего запроса от пользователя и отправляется на все исполняющие узлы сегмента. Задача чтения состоит из 2 этапов: чтения данных x с диска и преобразование данных в пары ключ-значения для стадии Reduce. Reduce-задача подразумевает подготовку данных после чтения с диска, в виде пар ключ-значение, и распределение их по исполняющим узлам. Информация для стадии Reduce отправляется на управляющий узел для разумного распределения между исполняющими узлами. Сложность задачи определяется по количеству KV-данных. Задача FinalReduce является практически полной копией задачи Reduce, однако она создается на основе результатов стадии Reduce. Суть задачи – найти исполняющий узел n_m, который менее всего загружен, и отправить исходные данные для задачи. Затем создается задача FinalReduce и эмулируется ее выполнение. Задачи балансировки – задачи l_b, выполняющие распределение задач l между исполняющими узлами n_w, основываясь на алгоритме балансировки. В качестве сравнения, кроме описанного алгоритма, использовался алгоритм WeightRoundRobin [4]. Задача балансировки имеет общий подход для всех типов задач:

1. Получение количества задач L от исполняющего узла и текущей нагрузки L_0;
2. Распределение алгоритмом перечня задач L;
3. Отправка на исполняющие узлы данных x_b о распределении;
4. Ожидание подтверждения получения данных;
5. Отправка на исполняющие узлы сообщения об окончании балансировки.

Вычислительные эксперименты

В разработанной системе были проведены следующие эксперименты:

1. Тестирование распределения задач между GPU и CPU.
2. Тестирование алгоритма распределения нагрузки.
3. Тестирование алгоритма выявления оптимальной топологии кластера.

Цель первого эксперимента – подтвердить полезность использования GPU и корректность распределения задач между процессорами в рамках одного узла вычислительного кластера. В результате эксперимента мы должны получить данные, подтверждающие полезность комбинирования CPU и GPU. Данные, на основе которых будет строится результат, – это время выполнения набора задач. Выдвигается гипотеза, что при комбинировании CPU и GPU ожидается увеличение скорости обработки данных примерно в 2 раза.

В ходе эксперимента было выполнено:

1. Получение временных данных по скорости обработки информации CPU;
2. Получение временных данных по скорости обработки информации GPU;
3. Вычисление эффективного распределения набора задач между CPU и GPU;
4. Сравнение времени обработки данных 3 способами: CPU, GPU, комбинируя CPU и GPU.

Для теста была взята классическая задача BigData – подсчет количества слов в тексте. В качестве исходных данных x был взят текст из 1000 строк, которые подразумеваются как перечень задач L, с примерно равным числом символов (около 200 символов). Для расчетов производилась только стадия Map.

После запуска выполнения задач L, распределенных между процессорами, был получен следующий результат: графический процессор значительно ускоряет обработку задач BigData в случае большого числа задач. При малом количестве задач GPU (в количестве около 200) уступает CPU, и при распределении задач расхождение очень велико ($|L_g| << |L_c|$). При уровне задач,

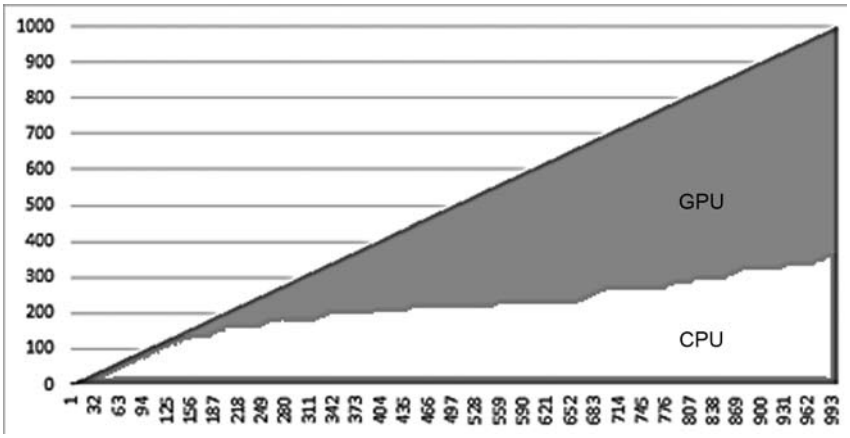

Рис. 5. Соотношение задач между CPU и GPU

близком к 1000, соотношение задач примерно 1:3, как показано на рисунке 5 (нижняя область – CPU, верхняя – GPU), и при увеличении количества расхождение будет расти. То есть эксперимент можно считать успешным.

Цель второго эксперимента – доказать эффективность работы алгоритма распределения задач L между управляющими узлами. Разработанный алгоритм сравнивался с алгоритмом WeightRoundRobin [4].

В качестве исходных данных эксперимента создается модель кластера DC, в котором 30 узлов, 3 сегмента и 3 управляющих узла, по каждому на сегмент. Характеристики скорости вычислений задаются в соответствии с данными, полученными при эксперименте распределения задач между CPU и GPU. Объем памяти задается равным 16 GB, скорость чтения с жесткого диска 450 Mb/s. Моделирование проводилось для потока задач L_{in} размерностью в 8000, 12000 и 15000 элементов.

Процесс моделирования запускается по очереди для 3 уровней потоков задач L_{in}, и выявляются нагруженные элементы в кластере.

В результате работы модели были получены следующие результаты, отображенные в таблице 1. Данные отражают количество загруженных узлов при равном количестве внешних запросов.

Таблица 1

Количество загруженных узлов

Запросов в секунду	WeightRoundRobin	Описанный алгоритм
8000	2	1
12000	5	3
15000	9	6

Как видно из таблицы 1, реализованный алгоритм работает эффективно, так как количество перегруженных исполняющих узлов Nw при использовании алгоритма WRR выше.

Цель последнего эксперимента – тестирование различных топологий. По очереди в системе производилось моделирование работы кластера DC, построенного в топологиях «Звезда», «Двухуровневое дерево», «Трехуровневое дерево». В качестве исходных данных создается модель кластера, в котором 30 узлов, 3 сегмента и 3 управляющих узла, по каждому на сегмент. Характеристики скорости вычислений задаются в соответствии с данными, полученными при эксперименте распределения задач между CPU и GPU. Объем памяти задается равным 16 GB, скорость чтения с жесткого диска 450 Mb/s. Для каналов передачи данных устанавливалась пропускная способность 1 Gb/s.

В ходе эксперимента были получены следующие результаты.

При запуске моделирования работы кластера DC, построенного топологией «Звезда», центральный узел N_c был перегружен. Кроме того, уровень передачи данных был высоким, а уровень загруженности канала передачи данных C_c был около 100%.

При запуске моделирования работы кластера DC, построенного топологией «Двухуровневое дерево», коммуникаторы (switch) были перегружены. Кроме того, уровень передачи данных был высоким и близким к 100%.

При запуске моделирования работы кластера, построенного топологией «Трехуровневое дерево», уровень загруженности данными был недостаточно высоким, однако все исполняющие узлы были перегружены.

Было выявлено, что при невысоком количестве запросов топология «Звезда» будет работать эффективно, однако при повышении нагрузки может возникнуть отсутствие данных. При изменении топологии на «Двухуровневое дерево» уровень предельной нагрузки повысился, однако то же имеется ограничение. При топологии «Звезда», загруженности данными не возникло. Но в случае увеличения количества исполняющих узлов в кластере перегруженность может иметь место.

Таким образом, можно сделать вывод, что выбор топологии зависит от уровня загруженности кластера. При высоком уровне загруженности лучше всего справится топология «Трехуровневое дерево» из-за равномерности распределения элементов кластера DC. При сравнении с топологией «Hadoop», которая объединяет группы узлов N в сегменты, количество ситуаций перегрузок будет меньше, т. к. фреймворк будет отдавать приоритет операциям внутри сегмента [2]. Но при большом потоке задач возникает перегрузка сегментов, что в свою очередь приводит к межсегментному обмену. В итоге это приводит к отказу всего кластера. В отличие от топологии «Hadoop», перегрузка элементов «Трехуровневое дерево» не приведет к отказу системы.

Заключение

В данной работе описана имитационная модель кластера и разработанная система моделирования, с помощью которой можно выработать методику повышения эффективности обработки больших объемов данных. Предлагаемая модель является трехуровневой, с межуровневыми зависимостями. В ней рассматривается влияние на эффективность работы кластера трех факторов: топологии кластера, алгоритма распределения нагрузки и повышения скорости вычислений за счет использования графических процессоров. Разработанная система моделирования может быть полезна для начального тестирования возможностей проектируемого кластера, который предполагается использовать для обработки больших объемов данных.

СПИСОК ЛИТЕРАТУРЫ

1. Большие данные : матер. свободной энциклопедии Википедиа. – URL: https://ru.wikipedia.org/wiki/BigData (дата обращения: 28.07.2017).

2. Уайт Т. Hadoop: Подробное руководство. – СПб. : Питер, 2013. – 672 с.

3. Принципы работы с большими данными, парадигма MapReduce. Ч. 1. – URL: https://habrahabr.ru/company/dca/blog/267361/ (дата обращения: 28.07.2017).

4. WeightRoundRobin : матер. свободной энциклопедии Википедиа. – URL: https://en.wikipedia.org/wiki/Weighted_round_robin (дата обращения: 28.07.2017).

5. Ларионов В.С. Балансировка нагрузки веб-серверов // Молодежный научно-технический вестник. – 2016. – № 03. – URL: http://sntbul.bmstu.ru/file/out/836534 (дата обращения: 28.07.2017).

6. Петров Н.С. Особенности построения кластера распределённой системы сбора и обработки информации датчиков // Известия ЮФУ. Технические науки.– 2015. – № 4 (165). – URL: http://cyberleninka.ru/article/n/osobennosti-postroeniya-klastera-raspredelyonnoy-sistemy-sbora-i-obrabotki-informatsii-datchikov (дата обращения: 28.07.2017).

7. Повышение скорости обработки данных с помощью локальности данных в Hadoop. – URL: https://habrahabr.ru/company/mirantis_openstack/blog/219177/ (дата обращения: 28.07.2017).

8. Кочетов Ю.А., Кочетова Н.А. Задача балансировки нагрузки на серверы // Вестник НГУ. Сер. Информационные технологии. – 2013. – URL: http://cyberleninka.ru/article/n/zadacha-balansirovki-nagruzki-na-servery (дата обращения: 28.07.2017).

9. BigData – BuzzWords: WhatisMapReduce – Day 7 of 21. – URL: https://blog.sqlauthority.com/2013/10/09/big-data-buzz-words-what-is-mapreduce-day-7-of-21/ (дата обращения: 28.07.2017).

10. Hadoop + GPU: Boost performance of your big data project by 50x-200x?. – URL: http://www.networkworld.com/article/2167576/tech-primers/hadoop---gpu--boost-performance-of-your-big-data-project-by-50x-200x-.html (дата обращения: 28.07.2017).

REFERENCES

1. Bolshie dannye. Mater. svobodnoi entsiklopedii Vikipedia [Big Data. Information from the Free Encyclopedia Wikipedia]. Available: https://ru.wikipedia.org/wiki/BigData (accessed: 28.07.2017).

2. White T. *Hadoop: Podrobnoe rukovodstvo* [Hadoop: The Definitive Guide]. St.Petersburg, Piter Publ., 2013. 672 p.

3. Printsipy raboty s bolshimi dannymi, paradigma MapReduce. Ch. 1 [Principles for Working with Big Data, MapReduce Paradigm. Part 1]. Available at: https://habrahabr.ru/company/dca/blog/267361/ (accessed: 28.07.2017).

4. WeightRoundRobin. Mater. svobodnoi entsiklopedii Vikipedia [WeightRoundRobin. Information from the Free Encyclopedia Wikipedia]. Available at: https://en.wikipedia.org/wiki/Weighted_round_robin (accessed: 28.07.2017).

5. Larionov V.S. Balansirovka nagruzki veb-serverov [Web-server Load Balancing]. *Molodezhnyi nauchno-tekhnicheskii vestnik* [Youth Scientific and Technical Bulletin], 2016, no. 03. Available at: http://sntbul.bmstu.ru/file/out/836534 (accessed: 28.07.2017).

6. Petrov N.S. Osobennosti postroeniia klastera raspredelennoi sistemy sbora i obrabotki informatsii datchikov [Features of Creation of the Cluster of the Distributed System of Data Collection and Processing of Sensors]. *Izvestiia IuFU. Tekhnicheskie nauki* [Izvestiya of SFedU. Engineering Sciences], 2015, no. 4 (165). Available at: http://cyberleninka.ru/article/n/osobennosti-postroeniya-klastera-raspredelyonnoy-sistemy-sbora-i-obrabotki-informatsii-datchikov (accessed: 28.07.2017).

7. Povyshenie skorosti obrabotki dannykh s pomoshchiu lokalnosti dannykh v Hadoop [Improving Data Processing Performance with Hadoop Data Locality]. Available at: https://habrahabr.ru/company/mirantis_openstack/blog/219177/ (accessed: 28.07.2017).

8. Kochetov Iu.A., Kochetova N.A. Zadacha balansirovki nagruzki na servery [The Servers Load Balancing Problem]. *Vestnik NGU. Ser. Informatsionnye tekhnologii* [Novosibirsk State University Journal of Information Technologies], 2013. Available at: http://cyberleninka.ru/article/n/zadacha-balansirovki-nagruzki-na-servery (accessed: 28.07.2017).

9. BigData – BuzzWords: WhatisMapReduce – Day 7 of 21. Available at: https://blog.sqlauthority.com/2013/10/09/big-data-buzz-words-what-is-mapreduce-day-7-of-21/ (accessed – 28.07.2017).

10. Hadoop + GPU: *Boost performance of your big data project by 50x-200x?.* Available at: http://www.networkworld.com/article/2167576/tech-primers/hadoop---gpu--boost-performance-of-your-big-data-project-by-50x-200x-.html (accessed: 28.07.2017).