Инженерный вестник Дона, №2, ч2 (2015)
ivdon.ru/ru/magazine/archive/n2p2y2015/3031

Сокращение времени оценки схожести текстовых документов на неоднородной многопроцессорной вычислительной системе

С.С. Серов, А.Е.Андреев, П.Д. Кравченя, Р.И. Гущин, П.П. Чеботарев

Волгоградский государственный технический университет

Аннотация: В работе рассматривается параллельная реализация упрощенного алгоритма шинглов для сокращения времени сравнения текстовых документов на неоднородной вычислительной системе на базе многоядерных процессоров и Many Integrated Core (MIC) ускорителей. Показана поэтапная модификация исходной однопоточной программы, рассмотрена архитектура распределенной программной системы для сравнения документов. Приведены результаты тестирования разработанных прототипов, показывающие возможность сокращения времени сравнения документов до 12 раз при использовании нескольких ускорителей в системе.
Ключевые слова: алгоритм шинглов, CRC32, Intel Xeon Phi™, MIC, OpenMP, Mono, ASP.NET

Введение

Задача определения степени схожести текстовых документов имеет достаточно большое практическое значение, прежде всего – для определения степени оригинальности заданного текста. Ее решают и поисковые системы Интернет, принимая решение об индексации или отказе от индексации новых ресурсов, и эксперты, оценивающие степень оригинальности учебных и научных работ. В последнем случае на помощь экспертам приходят различные системы антиплагиата. В частности, хорошо известна система Antiplagiat.ru [1], в течение ряда лет использующаяся в различных организациях. Для доступа к полным функциям системы, в частности, для поддержки своей базы работ, требуются определенные вложения, как на начальном этапе, так и на этапе поддержки лицензии. В связи с этим в разных учебных заведениях разрабатываются собственные системы, менее функциональные и менее надежные, но не требующие существенных вложений и имеющие при этом ограниченную полезность.

Одна из таких систем была разработана С.С. Серовым в 2012 году в рамках выпускной работы бакалавра и с тех пор достаточно успешно применяется на кафедре ЭВМ и систем факультета электроники и вычислительной техники (далее ФЭВТ) Волгоградского государственного технического университета (далее ВолгГТУ) в качестве вспомогательного средства оценки схожести студенческих работ.

Одним из положительных свойств подобных систем собственной разработки является возможность их модификации и развития в любом из интересующих авторов и пользователей направлении.

В данной работе предпринята попытка развития системы с точки зрения сокращения затрат времени на выполнение оценки текстов, а также – в направлении создания удаленного сервиса, использующего базу документов и возможности имеющегося вычислительного комплекса (кластера) ФЭВТ ВолгГТУ, оснащенного недавно новыми серверными платформами на базе процессоров Intel® Xeon® E5 v3 и ускорителей Intel® Xeon Phi™ [2].

Исходная система, разработанная в 2012 году, использовала известный алгоритм шинглов в упрощенной формулировке и была написана на языке C#, представляя собой настольное Windows-приложение с графическим интерфейсом пользователя (рис. 1).

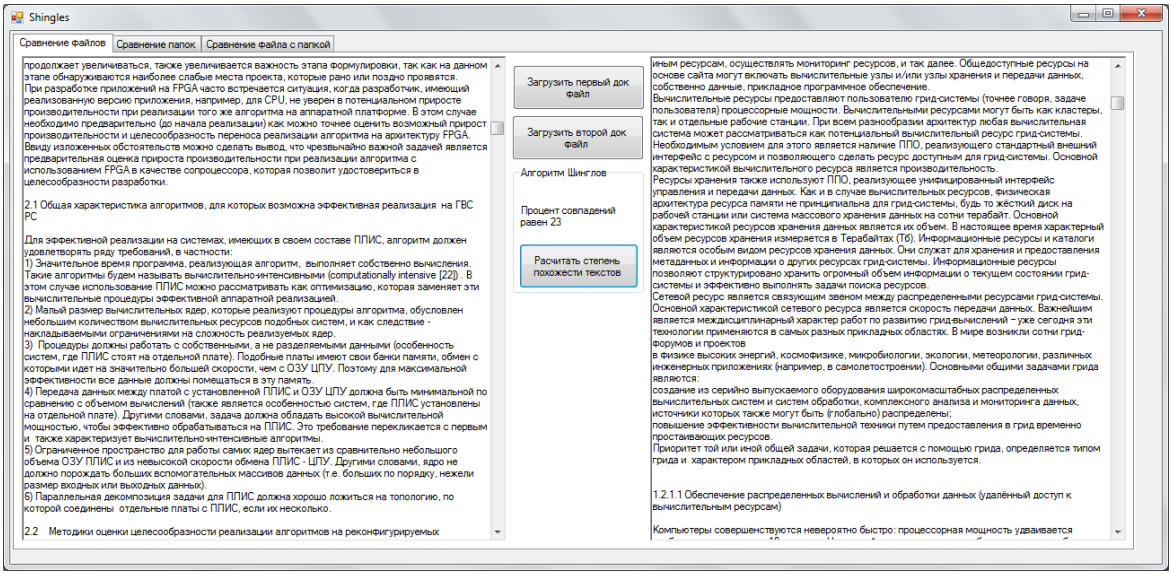

Рис. 1 – Внешний вид программы для оценки схожести файлов

Система оценивает степень схожести текстов и позволяет выявлять существенно похожие тексты. К недостаткам системы можно отнести низкую скорость работы, отсутствие возможности работы с базой документов (только сравнение документа с документами в заданном каталоге), возможность работы только с определенной версией документов Microsoft Word и другие.

Развитие системы предполагалось в следующих направлениях:

- использование многопоточных вычислений (в частности, на MIC архитектуре);
- создание сервиса для удаленного доступа, использующего серверное оборудование;
- создание клиента для предобработки документов в формате Microsoft Word, отправки заданий и отображения результатов;
- создание и использование базы имеющихся документов.

Выбор именно архитектуры Intel Many Integrated Cores – много интегрированных ядер (далее – MIC) обусловлен, в частности, доступностью большого числа MIC-ускорителей Intel® Xeon Phi™ в составе вычислительного кластера ВолгГТУ.

Упрощенный алгоритм шинглов

Описание алгоритма шинглов («чешуек») и его модификаций можно найти в литературе, в частности, в работах [3 - 5]. Алгоритм предполагает выделение в предварительно обработанном (канонизированном) тексте без предлогов и знаков препинания цепочек слов заданной длины (шинглов), расчет для каждого шингла контрольных сумм и затем подсчет числа совпадающих контрольных сумм в сравниваемых текстах. В упомянутой выше программе использован алгоритм шинглов в упрощенной постановке (рис.2).

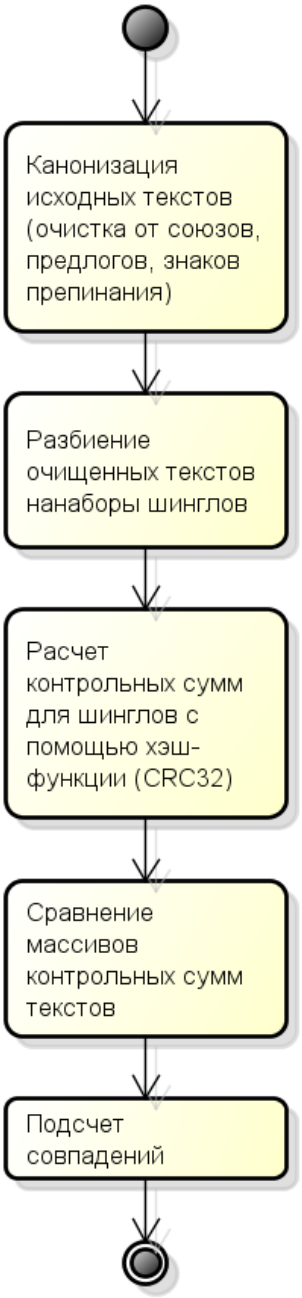

Рис. 2 - Схема упрощенного алгоритма шинглов

Варианты повышения быстродействия системы

Для системы можно выделить два варианта использования (прецедента):

- оценка степени схожести двух заданных текстовых документов;

- оценка степени схожести заданного текстового документа с имеющимися текстовыми документами.

Достаточно естественно, если второй вариант использования состоит в многократном применении первого. Однако, с другой стороны, если имеющиеся текстовые документы не изменяются, предпочтительнее хранить значения контрольных сумм для их шинглов и не рассчитывать их заново при обработке каждого нового документа. Тогда реализации этих вариантов использования будут отличаться.

Для повышения производительности при реализации обработки шинглов можно использовать разные подходы. При реализации обоих вариантов использования возможно использование параллелизма данных. Для второго прецедента сравнение каждой пары документов – совершенно автономная задача, соответственно, она может являться базовой подзадачей для каждого параллельно работающего вычислителя, потока или процесса. Для первого прецедента принципиально получение вначале списка шинглов и списка контрольных сумм для каждого шингла, а затем сравнение контрольных сумм. Базовыми подзадачами могут являться составление шингла, расчет его контрольной суммы и сравнение пары контрольных сумм.

В качестве возможного подхода для решения подобной задачи можно рассмотреть использование Map – Reduce [6]. В качестве этапа Map могут выступать составление шинглов и вычисление их контрольных сумм, затем может выполняться сравнение и группировка результатов на этапе Reduce. Для реализации этого подхода целесообразно использовать какой-либо из каркасов (фреймворков) Map-Reduce. В то же время такой подход потребует существенной переработки программы и самого алгоритма.

Другие подходы предполагают различные варианты реализации параллельной обработки и группировки базовых подзадач. Для рассматриваемой задачи возможно применение различных технологий

параллельных вычислений, включая OpenMP, OpenCL, CUDA [7], MPI и другие. Возможно применение многопоточных вычислений в рамках платформы .NET, на которой реализована исходная программа. Выбор технологии в данном случае в значительной степени определяется вычислительной платформой, которую предполагается использовать. Вариант с использованием Map-Reduce предполагает разворачивание соответствующей облачной вычислительной платформы, а также масштабируемые вычислительные ресурсы, эти вопросы в рамках небольшого вычислительного кластера ВолгГТУ пока не решены. Вариант с использованием массивно-параллельных графических ускорителей (GPU) (и соответственно – технологий OpenCL / CUDA) для данной задачи представляется достаточно эффективным [7], однако подобные вычислители доступны в рамках кластера ФЭВТ ВолгГТУ в небольшом количестве. Поскольку планируется перенос системы на неоднородную серверную платформу Intel Xeon E5 с ускорителями Intel MIC, в качестве основного был выбран вариант многопоточной реализации упрощенного алгоритма шинглов на языке C++ с возможным использованием API для Intel MIC [8].

Создание многопоточной реализации алгоритма

Поскольку было принято решение реализации многопоточной версии алгоритма на C++, исходная программа на C# была переписана на C++. Для тестирования производительности были подготовлены консольные приложения на C# и C++, которые в качестве параметров принимают имена двух сравниваемых текстовых файлов (работа с Microsoft Word в рамках прототипов не рассматривалась). В программе на C++ вместо стандартных не многопоточных коллекций использованы обычные одномерные массивы C.

В качестве базовой технологии многопоточности использована технология OpenMP. Для первого варианта использования необходимо было

на первом этапе алгоритма параллельно строить шинглы и рассчитывать их контрольные суммы, на втором – параллельно выполнять сравнение контрольных сумм (хэшей).

Для внедрения многопоточности использована стандартная прагма OpenMP `#pragma omp for`. Для цикла получения контрольных сумм шинглов :

```
#pragma omp parallel for
for (int i = 0; i <= (Words.size() - count_shingle); i++)
{
      string ShingleText = "";
     for (int j = 0; j < count_shingle; j++)
      {
          ShingleText += (Words[i + j] + " ");
      }
      int     res     =     crc.FullCRC((const     unsigned
            char*)ShingleText.c_str(),
            (size_t)ShingleText.length());
      buf[i] = res;
}
```

Для цикла сравнения контрольных сумм (применена также опция reduction) :

```
#pragma omp parallel for
for (int i=0; i < sizeA ; i++)
{
     for (int j = 0; j < sizeB; j++) {
          if ( bufB[j] == bufA[i] )
          { arrMatches[i]++; break; }
     }
}

matches = 0;
#pragma omp parallel for reduction(+:matches)
     for (int i = 0; i < sizeA; i++)
          matches += arrMatches[i];
```

Поскольку простое внедрение многопоточности в этап расчета контрольных сумм дало лишь незначительное ускорение, наряду с параллелизмом данных также был использован параллелизм задач –

параллельное вычисление контрольных сумм для обоих сравниваемых файлов с помощью прагмы `#pragma parallel section :`

```
#pragma omp parallel sections
{
     #pragma omp section
     {
          sizeA = GetShingles(Canonize(TextA), ShingleA);
}
     #pragma omp section
     {
          sizeB = GetShingles(Canonize(TextB), ShingleB);
     }
}
```

В таблице № 1 представлены результаты тестирования разработанных прототипов для трех пар документов на ноутбуке с процессором Core i5. Документы из первой пары содержат приблизительно 6 тысяч слов, второй пары – 20 тысяч слов, третьей – около 80 тысяч слов. Первая пара соответствует объему курсовой или семестровой работы, реферата, статьи, вторая пара соответствует объему выпускной работы, третья – диссертационной работы или монографии. Использовались документы с небольшим числом совпадений, т.к. их обработка по алгоритму занимает больше времени.

Таблица № 1

Время оценки схожести двух документов на ноутбуке с CPU Core i5

Объем документов (слов)	Однопоточные вычисления на CPU (C#), с	Однопоточные вычисления на CPU (C++), с	Ускорение	Многопоточные вычисления на CPU (C++, 2/4 потока), с	Ускорение
6 000	0.06	0.11	0.57	0.12	0.875
20 000	0.97	0.81	1.20	0.53	1.53
85 000	4.38	3.02	1.45	1.78	1.7

По результатам видно, что реализация на C++, как и можно предположить, дает ускорение для средних и больших документов, а многопоточная реализация дает ускорение на 2 ядрах до 1.7, но тоже только для объемных документов.

В таблице № 2 приведены результаты для однопоточной и многопоточной реализации, полученные для серверной платформы с использованием компилятора Intel C++.

Таблица № 2

Время оценки схожести двух документов на серверной платформе Xeon E5 v3

Объем документов (слов)	Однопоточные вычисления на CPU (C++), с	Ускорение по сравнению с Core i5	Многопоточные вычисления на CPU (C++, 20 потоков), с	Ускорение
6 000	0.06	1.75	0.11	0.55
20 000	0.51	1.59	0.20	2.55
85 000	1.88	1.61	0.47	4

Достигнуто ускорение в 4 раза на многоядерной платформе, или в 6.4 раза по сравнению с однопоточной реализацией на ПК. Одновременно можно заключить, что для небольших по объему документов, объемом менее 10 тыс. слов, многопоточный расчет не дает эффекта, соответственно в алгоритме нужно предусмотреть соответствующую проверку и использовать многопоточность, только начиная с объема от 10 000 слов.

Реализация алгоритма на ускорителе с архитектурой MIC

Ускорители Xeon Phi с архитектурой MIC поддерживают три режима эксплуатации [8]:

- Offload (режим разгрузки для центрального процессора, в этом режиме происходит автоматическая пересылка данных на ускоритель и обратно);
- Native (режим запуска всей программы на ускорителе из его файловой системы, т.к. ускоритель представляет собой отдельную вычислительную систему под управлением своей операционной системы; в этом режиме требуется явная пересылка и данных, и программы на ускоритель в виде файлов);
- Symmetric (поддержка одновременных вычислений на MIC и CPU, сложен в реализации).

Последний режим редко используется из-за своей сложности. Первый режим для ускорителей является преобладающим, но для следующего поколения устройств архитектуры MIC основным будет являться режим Native. В рамках данной работы использовались оба режима, при этом Offload показывал лучшие результаты из-за меньших накладных расходов на передачу данных на ускоритель. Поскольку первый этап алгоритма для первого прецедента (формирование шинглов и контрольных сумм для них) в данной реализации плохо поддерживает многопоточность, имело смысл реализовывать на 224-поточном MIC (с 57 ядрами) только сравнение контрольных сумм. Также необходимо отметить, что максимум производительности ускорителя можно получить только, если использовать его векторные 512-битные регистры. В данном случае есть возможность использовать 16-элементные векторы, так как сравниваются 32-битные коды. С помощью автоматической векторизации компилятором этого добиться не удалось, поэтому были использованы интринсики (С-обертки для векторных инструкций ускорителя) [9] для явного управления векторизацией :

```
#pragma omp parallel for
for (int i = 0; i < sizeA; i++)  {
     int A[16] __attribute__((aligned(64)));
     for (int j = 0; j < 16; j++) { A[j] = bufA[i]; }
     __m512i first = _mm512_load_epi32(A);

     for(int j = 0; j < sizeB; j += 16) {
          __m512i second = _mm512_load_epi32(&bufB[j]);
          __mmask16  found  =  _mm512_cmpeq_epi32_mask(first,
                    second);
          if( found ) { arrMatches[i]++; break; }
     }
}
```

В результате тестирования данного решения (расчет контрольных сумм на двух CPU Xeon E5 и сравнение их на Xeon Phi в режиме offload) при сравнении больших документов (третья пара) время вычислений составило 1.08 секунды, что примерно вдвое хуже, чем время сравнения только на двух CPU Xeon E5. Таким образом, применение MIC для сравнения документов размером до 100 000 слов нецелесообразно. Для реализации быстрого сравнения двух произвольных документов с расчетом контрольных сумм шинглов по CRC32 или другим алгоритмам можно рассмотреть применение других типов ускорителей, например, GPU или ПЛИС [10].

Был отдельно рассмотрен только этап сравнения хэшей на MIC, как наиболее удачный с точки зрения многопоточной реализации. Использовался режим offload. Тестирование проводилось на сгенерированных таблицах хэшей размером до 1000 000 значений. Результаты представлены в таблице № 3.

Время многопоточных вычислений для CPU несколько отличается от результатов из таблицы № 2, поскольку использованы другие входные

данные, в частности, для теста со 100 000 хэшей было много совпадений. Из таблицы 3 видно, что при увеличении объема сравниваемых таблиц MIC ускоритель начинает опережать CPU. На практике при сравнении двух файлов такой объем таблицы хэшей никогда не будет встречаться, он соответствует документам в несколько тысяч страниц. В то же время полученный результат позволяет предположить, что для второго варианта использования (сравнение документа с большим количеством документов, для которых заранее рассчитаны контрольные суммы) применение MIC также может быть эффективным.

Таблица № 3

Время сравнения двух таблиц контрольных сумм на серверной платформе Xeon E5 v3 с использованием и без использования MIC - ускорителя

Объем таблиц (хэшей)	Многопоточные вычисления на CPU (20 потоков), с	Многопоточные вычисления на MIC (224 потока), с	Ускорение
1000	0,06	0,72	0,08
10 000	0,07	0,68	0,1
100 000	0,41	0,81	0,51
1000 000	3,51	2,59	1,36

При решении задачи сравнения заданного файла со списком уже загруженных в систему файлов применение MIC должно позволить получить потенциально ускорение пропорционально количеству имеющихся в системе ускорителей. В этой задаче нет необходимости вычислять хэш-функции шинглов для ранее загруженных файлов и все вычисления сводятся к сравнению значений хэшей шинглов заданного файла с ранее загруженными значениями хэшей файлов из списка, распределенными между разными ускорителями. Был разработан прототип для сравнения 10 000 хэшей с 16000 хешами в 100, 200 и 300 файлах на двух CPU и на одном MIC ускорителе в

режиме Native (данные для обработки предварительно должны быть загружены в файловую систему MIC). Результаты тестирования прототипа показаны в таблице № 4.

Таблица № 4

Время сравнения заданной таблицы контрольных сумм (10000 значений) с большим числом таблиц контрольных сумм на двух CPU Xeon E5 v3 и на MIC - ускорителе

Количество таблиц хэшей	Многопоточные вычисления на CPU (20 потоков), с	Многопоточные вычисления на MIC (224 потока), с	Ускорение
100	0,58	0,54	1,07
200	1,09	0,60	1,81
300	1,56	0,80	1,95

При количестве файлов от 200 и выше время сравнения на одном ускорителе MIC примерно в два раза больше, чем на двух CPU. Объем файла с хэшами при этом не превышает 30 Мб, такой файл можно спокойно разместить в файловой системе ускорителя, объем локальной памяти которого 8Гб. Полученное соотношение производительности MIC ускорителя к CPU Xeon E5 v3 (2 к 1) для данной задачи близко к соотношению величин их пиковой теоретической производительности (1 к 0.7) для операций с плавающей запятой. Это стало возможным благодаря использованию 512-битных регистров MIC с одновременной обработкой 16 целочисленных элементов векторов.

Архитектура сервиса оценивания схожести текстов

В результате архитектура сервиса, использующего неоднородную вычислительную систему на базе Xeon E5 и нескольких Xeon Phi для оценивания схожести заданного текста с имеющимся набором текстов, может выглядеть так, как представлено на рис. 3. При этом для первого варианта

использования сервиса (сравнение двух заданных текстов) будут использоваться только центральные процессоры, причем до определенного объема файлов – в однопоточном режиме. Для второго варианта использования на этапе сравнения будут задействованы ускорители, в файловой системе которых будет распределена база ранее посчитанных хэшей имеющихся файлов. В качестве обертки для C++приложений, работающих под операционной системой Linux, может использоваться ASP.NET веб-сервис, реализованный на платформе Mono, либо Java веб-сервис на платформе Tomcat.

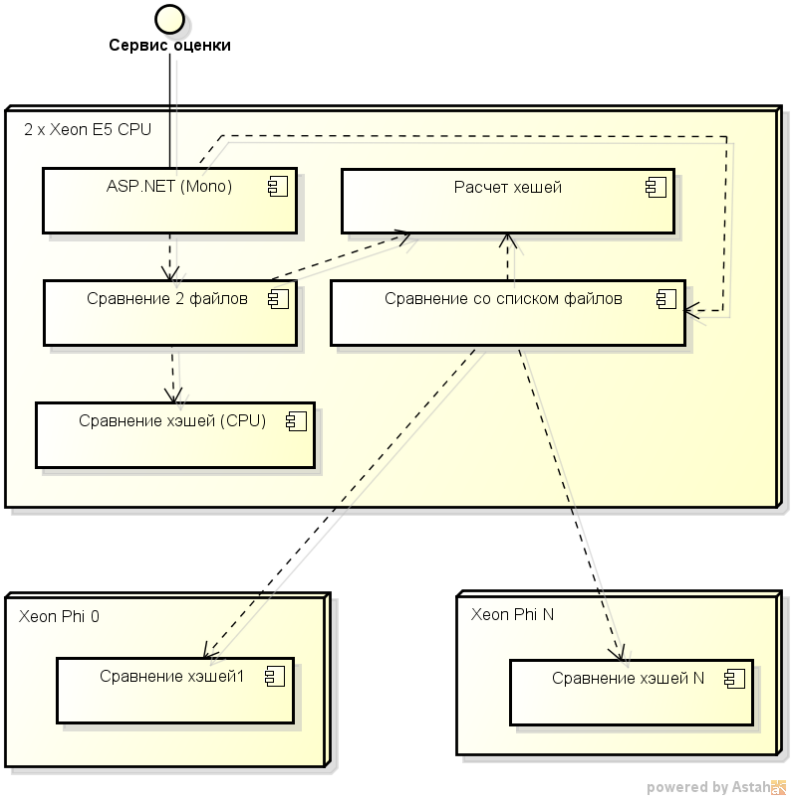

Рис. 3 – Архитектура веб-сервиса

Выводы

Разработана параллельная реализация упрощенного алгоритма оценки схожести текстов для неоднородной системы на базе CPU Xeon E5 и MIC-ускорителей Xeon Phi. Предложена архитектура веб-сервиса. Показано, что

для одной серверной платформы с 6 ускорителями (такая платформа имеется в составе кластера, а в целом на рынке имеются платформы с возможностью размещения 8 ускорителей) можно получить ускорение сравнения файлов со списком имеющихся до 12 раз по сравнению с двумя CPU Xeon E5 v3, до 48 раз по сравнению с однопоточной реализацией на этих же CPU и до 70 раз по сравнению с использованием обычного ПК. Задача сравнения двух произвольных файлов решается с помощью центральных процессоров сервера и за счет внедрения многопоточности ускоряется до 4 раз по сравнению с одним потоком на сервере и до 6.5 раз по сравнению с одним потоком на ПК для файлов большого объема.

Литература

1. Антиплагиат– http://www.antiplagiat.ru/.
2. ТОП 50. Суперкомпьютеры. Текущий рейтинг. 22-ая редакция от 31.03.2015. - http://top50.supercomputers.ru/?page=rating.
3. Broder A., Glassman S., Manasse M. Zweig G. Syntactic Clustering of the Web // Comput. Netw. ISDN Syst. 1997. Vol. 29. pp. 1157–1166
4. Зеленков Ю. Г., Сегалович И.В. Сравнительный анализ методов определения нечетких дубликатов для Web–документов // Тр. 9-й Всеросс. научной конф. «Электронные библиотеки: перспективные методы и технологии, электронные коллекции». — Переславль-Залесский: Изд-во ИПС РАН, 2007. С. 166–174.
5. Алгоритм шинглов для веб-докум ентов, поиск нечетких дубликатов текста, сравнение текстов на схожесть. http://www.codeisart.ru/part-1-shingles-algorithm-for-web-documents/.
6. Янишевская А.Г., Чурсин М.А. Способы хранения и обработки большого объема данных с использованием MapReduce и Percona Server // Инженерный вестник Дона, 2015, №2, ч.2 URL: http://ivdon.ru/ru/magazine/archive/n2py2015/2976
7. Солошенко А.Н., Орлова Ю.А., Розалиев В.Л., Заболеева-Зотова А.В. Establishing Semantic Similarity of the Cluster Documents and Extracting Key Entities in the

Problem of the Semantic Analysis of News Texts // Modern Applied Science, 2015. - Vol. 9, No. 5. - C. 246-268.

8. Intel® Xeon Phi™ Coprocessor System Software Developers Guide. software.intel.com/en-us/articles/intel-Xeon-Phi-coprocessor-system-software-developers-guide.

9. Intel® Intrinsics Guide: software.intel.com/en-us/articles/intel-intrinsics-guide.

10. Быков Д.В., Неретин А.Д. Прогнозирование производительности при реализации алгоритмов генерации случайных последовательностей больших размерностей на реконфигурируемых архитектурах с сопроцессорами // Инженерный вестник Дона, 2014, №2. URL : ivdon.ru/ru/magazine/archive/n2y2014/2414

References

1. Antiplagiat. http://www.antiplagiat.ru/.

2. TOP 50. Superkomp'jutery. Tekushhij rejting. 22-aja redakcija ot 31.03.2015. - http://top50.supercomputers.ru/?page=rating.

3. Broder A., Glassman S., Manasse M. Zweig G. Syntactic Clustering of the Web. Comput. Netw. ISDN Syst. 1997. Vol. 29. pp. 1157–1166

4. Zelenkov Ju. G., Segalovich I.V. Tr. 9-j Vseross. nauchnoj konf. «Jelektronnye biblioteki: perspektivnye metody i tehnologii, jelektronnye kollekcii». Pereslavl'-Zalesskij: Izd-vo IPS RAN, 2007. pp. 166–174.

5. Algoritm shinglov dlja veb-dokumentov, poisk nechetkih dublikatov teksta, sravnenie tekstov na shozhest'. http://www.codeisart.ru/part-1-shingles-algorithm-for-web-documents/ (dostup svobodnyj)

6. Janishevskaja A.G., Chursin M.A. Inzhenernyj vestnik Dona, (Rus), 2015, №2.ch.2 URL: ivdon.ru/ru/magazine/archive/n2py2015/2976

7. Soloshenko A.N., Orlova Ju.A., Rozaliev V.L., Zaboleeva-Zotova A.V. Modern Applied Science, 2015. Vol. 9, No. 5. pp. 246-268.

8. Intel® Xeon Phi™ Coprocessor System Software Developers Guide. software.intel.com/en-us/articles/intel-Xeon-Phi-coprocessor-system-software-developers-guide.

9. Intel® Intrinsics Guide. software.intel.com/en-us/articles/intel-intrinsics-guide (dostup svobodnyj).

10. Bykov D.V., Neretin A.D. Inzhenernyj vestnik Dona, (Rus), 2014, №2. URL: ivdon.ru/ru/magazine/archive/n2y2014/2414