

本科毕业论文（设计）

英文文献翻译

|  |  |
| --- | --- |
| **文献英文题目** | **Hbase - non SQL Database,Performance Evaluation** |
| **学 生 姓 名** | **吴光宇** |
| **学号** | **2014051016** |
| **专业** | **计算机科学与技术** |
| **年级班级** | **2014级计算机（应用）1班** |
| **指导教师** | **张欢（讲师）** |
| **所在学院** | **计算机学院** |
| **提交日期** | **2018年4月18日** |

2018 年 4 月

成都信息工程大学 计算机学院

英文文献题目

Dorin Carstoiu1，Elena Lepadatu2，Mihai Gaspar3

1，Politehnica” University of Bucharest, dorin.carstoiu@yahoo.com

2，3，Politehnica” University of Bucharest, lepadatu.elena@gmail.com, gaspar.mihai@yahoo.com

摘要：HBase是BigTable的开源版本-由google开发的用于管理大量结构化数据的分布式存储系统，HBase使用java进行开发，并完整的实现了BigTable所描述的功能，这篇文章的工作是评估HBase与传统SQL数据性能差异，当然BigTable的表现也应该大体相同。测试主要包括随机写入，顺序写入和顺序读出，以及增加列簇将会对它们产生什么影响，同时将测试使用MapReduce函数的随机读写性能。

关键词：HBase；BigTable；MapReduce；Hadoop；数据管理；namenode

# 介绍

一般来说，数据库是在外部介质上存储数据的一种方式。通常，数据库中的数据存储在多个文件中且从这些文件中再次将数据读取出来是可能的。使用数据库的往往是哪些管理庞大且复杂数据（数据操作）的系统，数据库对于管理这些数据（可能是密集的）是非常有效的。这也是在大型应用中使用数据库存储的主要目标。在数据库中的数据是数据库管理员精心设计过且满足范式的数据，所以这些数据具有以下一些优点：可以减少数据的冗余性、通过减少数据副本进而避免数据不一致性、共享数据、实施安全限制并为维护数据完整性和存储标准化提供了可能性。当然，使用方便也是十分重要的。

虽然在数据库中组织数据的方式有很多种，但是对于关系型数据库来说，使用一组数学理论来组织数据仍旧是最有效的方法之一。一个典型的数据库模型是关系数据库模型，根据这个模型，数据被存储在表中，表示行的集合，行是列的集合，列表示一个单独的属性。除了表外，数据库还包括：存储过程，视图，用户和用户组，数据对象类型以及函数。

集中式数据库的主要目标是：统一控制、最小化数据冗余、高效率以及同样重要的数据完整性和数据安全性。但是由于数据库技术和通信技术的提升，分布式数据库已经开始逐渐崭露头角。

分布式数据库在逻辑上是一个集成数据库，但是在物理上分布在通过网络等基础设施进行通信的多台物理机上。

关系型数据库的主要优点是数据的可访问性和检索信息的便捷性，但是这依赖于编写正确的SQL语言。

当数据量足够庞大的时候，关系型数据库将丧失他的优点，甚至变得不可使用，因此，关系型数据库渐渐被非关系型数据库取代，而诸多事实也证明如此：构建一个关系型的分布式数据库所需要的基础设施十分昂贵，且保持关系型数据库的正确属性（例如索引、完整性、主键和外键）将变得十分困难，这会使人十分困扰，但另一方面，因为关系型数据库查询方便，这使得它们在世界范围了被广泛使用。

现代的许多应用程序包括一个数据库服务器，它将被用来响应来自多个客户端的Web请求，在这种情况下，人们常常发现实际使用的性能低于预期的性能，为了解决这个问题，许多人考虑升级硬件，而不是考虑是否是数据服务器本身的问题。举例来说，运行在Oracle数据库上的Amazon.com无论是在优化还是扩展上都达到了SQL技术的顶点。

目前，存在很多使用数据库的应用程序，它们的性能瓶颈或多或少都与SQL组件有关，这些SQL组件往往以复杂的方式去执行非常简单的任务，这种方式适合80年代的计算机，但是不适合当前的体系结构。大部分大公司开发的数据库管理系统很大程度上需要依赖硬件来确保数据库的性能，当对这种数据库进行扩展的时候，成本将变得极其的高。所以现在急需一种新的方法，可以以较低的成本获得大幅的性能提升，并可以提供良好的可扩展性。

考虑其他方法是一种必然的结果，其中很好的一个例子就是Google的方法：将BigTable作为半结构化数据库来使用并将绝大多数internet信息存放在缓存中[3],通过比较两种方法，可以得出结论：传统SQL数据库，诸如 Oracle，DB2将不适应这类应用程序。与谷歌Bigtable相类似的方法是在80年代左右通过操作系统引入的所谓的“分层文件系统”。

非SQL数据库在2009年初开始表现的更加强势，当时他们提出了分布式数据库的解决方案，这些解决方案可用于不需要RDMS中存在关系特性的系统。正如前面提到的那样，分布式的非SQL数据库提供了人们十分需要的可伸缩特性。BigTable开创了非SQL数据库的市场，从2004年年初开始，它依赖于google的文件系统，支持对于大型信息数据集高吞吐量访问并提供了极高的可扩展性。BigTable和Cassandra、Hypertable、Hbase、Dynamo等开源项目都具有以下特性：键值存储，批量分析，大量机器间分区和共享数据，这些技术的另一个特点就是牺牲数据的强一致性来换取高可扩展性、可用性、性能和容错性。

# 相关工作

基于键值对存储的存储解决方案是一个不错的选择，理由主要有以下几点：

1. 许多关系型数据库并不能保证数据能够正确被复制，而可靠性比较高的关系型数据库又会花费大量的资金；
2. 存储大量半结构化数据的需求一直存在；
3. 可以很好的和新型语言结合，比如Erlang；
4. 数据通常基于主键存储和访问，在处理数据时无需复杂的连接操作；
5. 如果数据量过于庞大，处理因复制引起的错误将变得十分困难；

例如，Facebook使用了Haystack，它将大量数据存储在一个带有独立索引的文件中，而索引的开销很小，比如1G的数据只需要存储1M的元数据。现在已经开发出了一些替代RDBMS的产品，这些项目大都不仅仅只有一个关键值存储，还会具有以下一些特性：

1. 实施语言-用于Hypertable，ThruDB，MemcacheDB的Voldermort，Cassandra,Hbase，Erlang for Ringo,kai,Scalaris，Dynomite，C(C++)
2. 数据模型：主要是blob，面向文档或BigTable
3. 容错主要是基于复制和分区

其中一些提供基于块复制方式的分布式存储服务。另外一个重要的问题是数据用来填充动态页面的延迟，特别是针对web应用程序。延迟通常取决于实际环境和缓存中是否存在需要的数据，一般来说，获取数据的延迟不应该超过10毫秒，否则分析这些数据将花费更多的性能成本，一些流行的非SQL数据库实现是：

1. Google的BigTable[3]是一个分布式存储系统，用于管理高度扩展到结构化数据。该系统已经在Google的重要应用程序中证明论其效率。比如谷歌分析，谷歌地球，谷歌财经。BigTable为用户提供了一个使用行，列和时间戳的简单数据模型。从数据模型的角度来看，BigTable是一个稀疏的分布式的持久多维排序映射，其中映射中的每个值都是不可解释的字节数组。作为非SQL数据库的典型特征，列被分为称作列簇的结合，其通常包含相同类型的信息，时间戳是为每个单元可以包含该值的多个版本而引入的。
2. 非SQL数据库的另一个例子是Cassandra。这个系统的发展始于几年前的Facebook，目前这个项目仍是Apache Software Foundation下的一个开源项目。这个系统是Dynamo的分布式体系和BigTable提出的列族模型的混合体，从数据模型的角度来看，Cassandra是一个关键字索引的多维map，每个应用程序创建自己的键空间，除了列族之外，还介绍了一种用于表示列列表中不同列的新概念。数据在RandomWrite和SequentialWrite操作的列上表现良好。在每行中，列也按其名称进行排列。Cassandra提出的分区解决方案类似于BigTable提供的解决方案，比如使用散列函数。系统内部的持久性依赖于本地文件系统。
3. 另一个例子是LinkedLn的工程师推广的方案Voldemort[14]，该方案除了提供了与上述相同的特征，还提供了一些新的特征：序列化，支持只读节点，压缩。LinkedLn使用这个系统作为其底层存储系统。

# Hadoop框架

Hadoop构建非关系型世界的方法非常简单，提供了可靠的共享存储和分析系统[12]。存储由HDFS实现，而分析由MapReduce实现，虽然Hadoop还有其他功能，但是HDFS和MapReduce是最常使用的，因此也是最为重要的。

这种架构的主要优点是可以有效节省成本。它基础架构的核心思想在于如何存储PB级别的数据并提供共享和访问。即时使用多个磁盘写入和读取的想法会带来一些麻烦，但是这使得简单查询的时间得到极大的改善，所以一些缺点是可以接受的。比如不可避免的硬件故障：一旦你开始使用大量的硬件，单个硬件出现故障的概率是非常高的，而解决因硬件故障问题造成数据丢失的常见方案是在整个复制过程中，由系统保存数据的冗余副本，以便在发生故障的时候可以使用另一个副本。Hadoop通过为每个数据块保存3个副本的方式来处理这个问题。第二个问题是大多数分析任务需要以某种方式组合数据；从一个磁盘读取的数据可能需要与其它100个磁盘的数据进行组合，各种分布式系统运行从多个来源组合数据，但是要确保这项功能的正确性是极具挑战的。

MapReduce提供了一个编程模型，他将问题从磁盘的读写中抽象出来，并转化为键值对的组合问题。因为它提供了对Java、Python、C等语言的支持，所以很容易与经典的编程语言相结合。现在应该关注的是MapReduce两个重要的部分，Map和Reduce函数，当然，像map和reduce交互的部分也是存在并且也很重要的，与HDFS相同，MapReduece也提供了在数据存储和计算方面的可靠性。

如今，Hadoop是Apache Software Foundation旗下的顶级开源项目，Hadoop的一些其它组件，如Zookeeper、Core、Pig为Hadoop提供了研究大数据的能力。

MapReduce通常并行工作，从而使人们可以同时访问大量机器，从而获得对大量数据的处理能力，MapReduece的工作通常分为两个阶段：Map阶段和Reduce阶段。每个阶段都有键值对作为输入和输出，这些类型可以由程序员自己决定。程序员使用MapReduce编程模型需要编写两个函数，map函数和reduce函数，至于编写的语言，则随程序员自己的喜好。

MapRecue作业是集群工作的基本单位：它由输入数据，MapReduce程序和配置信息组成。MapReduce将作业分为任务来完成，任务有两种类型：map任务和reduce任务，MapReduce还提供了一个JobTracker和一些TaskTracker，JobTracker通过协调系统上运行的所有作业以及建立在TaskTracker运行任务的时间表来协调工作，TaskTracker负责运行任务同时收集任务的运行信息并将这些信息发送给TaskTracker，需要重点说明的是，如果一个任务失败，那么TaskTracker将在另外的节点重启这个任务[15]。

MapReduce作业通常将输入分为固定大小的部分，这个过程称为输入拆分，Hadoop为每个任务创建一个Map作业，用户需要为这个作业定义函数或者过程。

粗略得想，对于Hadoop来说运行Map/Reduce最好的节点选择就是数据所在的那些节点，这个概念通常被称作数据的局部性，这是数据检索中非常重要的一个特性。

Map任务将其输出写入本地磁盘，而不是HDFS。Map过程的输出是中间输出，通过Reduce过程处理的数据才是最后结果，一旦作业完成，Map的输出可以被丢弃，因此，将这些数据存入HDFS将是一件得不偿失的事情。如果Map过程在被Reduece过程使用前执行失败，Hadoop将在另外一个节点重新执行该任务以重建Map过程的输出。

当数据量大于单个物理机存储的极限，HDFS就派上了用场，我们可以使用几台物理机来处理这些数据。通过机器网络管理存储数据的文件系统统称为分布式文件系统，由于它们是基于网络的，网络编程的所有复杂性都可能出现，从而使分布式文件系统比常规磁盘系统都更为复杂，这个领域最有挑战性的问题就是如何处理单节点故障导致的数据丢失问题。

块是HDFS文件系统读取和写入文件的基本单位，但是通常块都比较大，默认值通常是64MB，Hadoop存储数据的方式和单个磁盘存储数据的方式相同：文件被分解为块大小数据，它们将被作为独立单元存储。使用基于块结构的组织方式对于分布式文件系统来说具有很多好处。第一个好处也是最明显的一个好处：一个文件可以比网络中的任何一个磁盘都大，其次，使用块的工作单元简化了存储子系统。简单是所有数据库开发人员共同期望的，特别是在经常发生节点故障的情况下。由块组成的存储子系统简化了存储管理（因为块大小是固定的，所以更容易估计磁盘的维度），并且消除了一些元数据可能存在的问题[13]。

HDFS集群中有两种节点：Namenode（主节点）和Datanode（从节点），Namenode负责管理文件系统的命名空间，它维护描述目录和文件树的元数据。Namespace镜像文件和操作日志将以更为持久的方式记录这些信息。Namenode可以查询出表示给定文件的块所在的Datanode，但是它不会永久的存储这些信息，因为这些信息将会在每次重启的过程中从datanodes重建。对文件系统的访问是通过用户与namenode和datanode进行通信实现的[12]。

Datanode是文件系统中真正用来存储数据的节点，它们负责处理客户端或namenode发出的存储请求并定期向namenode发送自己维护的块列表。

当丢失了namenode，文件系统将变得不可使用。事实上，如果运行namenode的机器被关闭，那么文件系统上的所有文件都将丢失，因为没有办法通过namenode获取存储文件的块的datanode信息。这就是为什么namenode具有容错性十分重要的原因，Hadoop为此提出了两个解决方案，第一个是备份组成文件系统元数据持久状态的文件，可以通过配置Hadoop，以便namenode可以将这些文件写入多个文件系统，这些写入都是同步且原子性的。

HBase是建立在HDFS之上的分布式列数据库。当我们需要实现读/写/随机访问大量数据集的时候，HBase就诞生了。虽然我们可能会有寄希望于关系型数据块来解决这个问题，但是它们并不是以大规模和分布式来建立的。

许多供应商试图找到解决这个问题的方法，因此提出了不同的解决方案。他们提供通过复制和分区的方式来扩展数据块，使其超越了单个节点的限制，但是这些附加组件只是迫不得已的选择，安装和维护通常也十分麻烦。

这些解决方案对RDBMS功能集的支持也不尽如人意，当我们谈论关于RDBMS连接的扩展时，维护复杂的查询、触发器、视图和外键约束的代价将变得极其高昂。HBase另辟蹊径来解决这个问题，只需要添加节点就可以从零开始构建一个HBase集群。HBase第一个用途就是存储web表格，一个由爬虫抓取的web信息。

使用MapReduece的应用程序将数据存储在表中，表由行和列组成，表单元格由不同的版本，默认情况下，版本通过每次在单元格中插入数据的时间戳来表示，通常情况下，它将被存储为字节数组。

行键也是字节数组，所以理论上任何东西都可以是行键，不管是字符串还是二进制表示的长键甚至是序列化的数据。行键按字典序进行排序，所有访问表的操作都通过行键来实现[9]。

即时我们不讨论在非SQL数据库中使用索引，但还是要指出HBase使用列族作为对关系索引的响应，因此列被分组到列簇中，所有列簇成员有一个共同的前缀。

HBase自动将数据分配到不同region，每个region都包含一个表的子集，一个region由其第一行（含）和最后一行（不含）加上一个随机生成的区域标识符来定义。就像HDFS和MapReduce由运行在其中的节点组成一样。

HBase的特征还在于namenode负责管理所有的datanode节点，HBase主要负责初始安装并为datanode划分region以及恢复服务器故障[4]。Namenode并没有被真正加载，因为namenode并不会存储数据。

一个region server会管理多个region，并响应来自客户端的读写请求。

Region server通过HBase conf/regionservers进行配置，就像在Hadoop conf/slaves文件中列出数据节点一样，启动和停止脚本与hadoop中使用SSH远程命令机制相同。Conf/hbase-site.xml和conf/hbase-env.sh文件用于对集群的配置，格式与HDFS配置相同。

HBase也有一些名为-ROOT包含.META的列表。.MATA表中包含所有用户空间区域的列表。

# 系统架构

用于测试的硬件平台由两台通过路由器连接在一起的机器组成：一台台式机和一台笔记本电脑，台式机配置是：AMD Sempron 2600+ 1.6 GHz，768 MB DDR RAM，160 GB硬盘，操作系统Windows XP SP2 32位，笔记本配置是：Intel Core 2 Duo T5550 1.83 GHz，2 GB DDR RAM，160 GB硬盘，安装Windows 64位操作系统。

有两种安装hadoop平台和hbase的方式：在windows上使用linux环境的软件模拟安装，或者直接在Linux的系统上进行安装。本次测试选择的是使用VMware Player安装linux虚拟机，台式和笔记本各安装一个。

笔记本安装的虚拟机代表集群的主节点（网络的第一个节点-笔记本），该虚拟机配置为：524 MB的RAM，单个处理器，9 GB硬盘，一个网络适配器。

两台虚拟机安装的操作系统都是ubuntu 9.10 32位。安装的jdk版本为Sun Java 1.6.0\_20，并创建了一个用来运行Hadoop和HBase的用户，安装了SSH服务器并创建了SSH密匙，这样两台电脑可以免密登陆，然后启动hadoop平台安装。

从hadoop平台的网站上下载最新版本的hadoop，并修改hadoop-env.sh,接着通过core-site,hdfs-site.xml和mapred-site.xml对hadoop进行配置。

在core-site.xml文件中设置了默认hadoop文件系统的名称，在hdfs-site.xml文件中设置文件系统的位置，并设置HDFS中数据保存的副本数。

Mapred-site.xml设置mapreduce作业JobTracker的主机和端口，在配置好相关文件后，需要对HDFS文件系统进行初始化，需要说明的是，应该在两个节点维护相同的配置。

从同一个站点下载最新半杯的HBase。其操作和hadoop类似，只是配置文件为hbase-env.sh，其中需要配置JAVA\_HOME和禁用IPv6。

在HBase-site.xml中修改了4个参数：HBase的根文件夹，HBase运行模式（完全分布式），以及Zookeeper的位置和名称，同样的，这些文件应该在两台电脑上保持同步。

# 性能评估

## 表行数影响

HBase按行ID以升序存储表中的记录，这个测试的目的在于测试HBase连续和随机读取行的性能。使用10.000,100.000，300.000，500.000，700.000和100.000来评估HBase性能。

对于上述每种情况，按照以下几个步骤进行：

第一步：创建一个名为“test”的表，使用单个列簇和单个列，然后插入随机生成的包含1000字节值的行（按行ID随机插入）。

第二步：删除第一步中创建的表格，再创建一个名为“test”的表，使用单个列簇和单个列，然后插入随机生成的包含1000字节值的行（按行ID顺序插入）。

第三步：使用第二步的表格，对表中数据进行顺序读取。

第四步：使用第二步的表格，对表中数据进行随机读取。

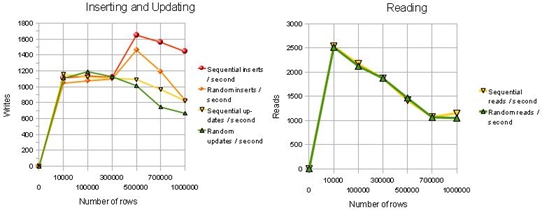
第五步：使用第二步的表格，对表中数据进行顺序更新。

第六步：使用第二步的表格，对表中数据进行随机更新。

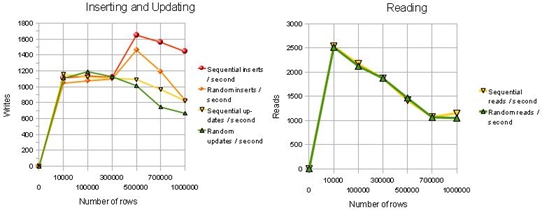
**Table 1.** Testing by number of rows

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Number of rows** | **10000** | **100000** | **300000** | **500000** | **700000** | **1000000** |
| Sequential inserts / second | 1112 | 1136 | 1119 | 1650 | 1560 | 1443 |
| Random inserts / second | 1045 | 1075 | 1098 | 1461 | 1194 | 832 |
| Sequential readings / second | 2534 | 2173 | 1875 | 1418 | 1075 | 1154 |
| Random readings / second | 3519 | 2127 | 1875 | 1466 | 1069 | 1054 |
| Sequential updates / second | 1154 | 1123 | 1110 | 1091 | 967 | 824 |
| Random updates / second | 1107 | 1190 | 1127 | 1014 | 745 | 666 |

Hbase - non SQL database, performances evaluation



Dorin Carstoiu, Elena Lepadatu, Mihai Gaspar



**Figure 1.** Inserting, updating and reading records

从图1中可以看出，随机插入的性能与顺序插入的性能差距不大，写入速度在5000行的时候达到峰值，之后随着行数的递增插入速度略微下降，对于更新操作，在处理300000行数据的时候比直接插入数据效率更高，之后随着行数的增加，数据更新的性能有所下降，但是如果表的数据量很大，更新速度将不会有太大的波动。

与google的bigtable相比，hbase执行这六个操作的效率远低于在bigtable上执行相同操作的效率，bigtable使用单个服务器连续读取的平均速度是每秒4000次。造成这种情况的原因之一可能是这个测试使用的硬件资源低于google使用的硬件资源。

## 列簇数的影响

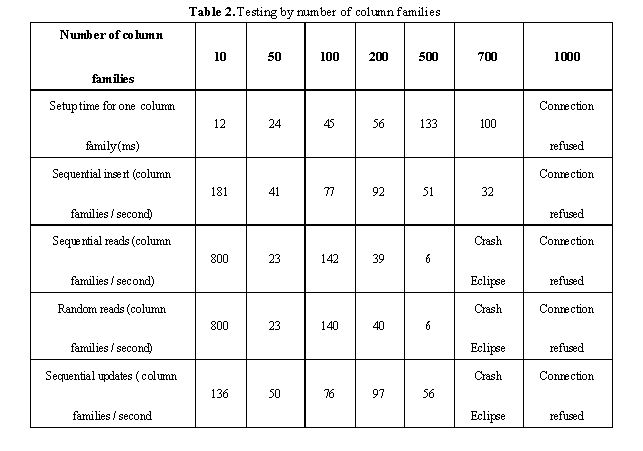
列簇对于hbase性能的影响是一个值得讨论的问题。在这个测试中，我们研究了在拥有多个列簇的表中进行读写和更新的效率，并且试图确定在hbase表中可以使用的列簇的最大数量[8]，这个测试的列簇数量将从相对较少的列簇到更大的1000个列簇。

使用拥有10,50,100,200,500,700和1000列簇的表评估hbase的性能，具体步骤如下：

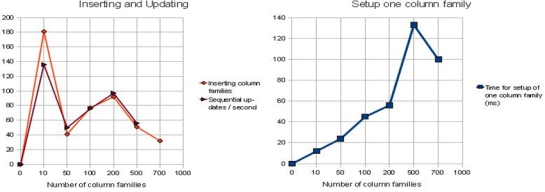
第一步：创建test表，表中只有一行，但是有多个列族，每个列族只有一个列，然后将随机生成的1000字节数据顺序插入到表中。

第二步：使用第一步中的表，连续顺序读取5000个数据。

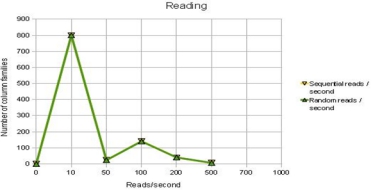
第三步：使用第一步中的表，连续随机读取5000个数据。

第四步：使用第一步中的表，对表格中的行进行顺序更新

上表显示当列簇数量过多的时候，增加一个列族消耗的时间将非常的多。



记录和更新的效率相差无几，但是在列族大于200的时候，记录和更新的效率显著下降。当列族的数量超过700的时候，唯一能做的就是建立表和插入数据，任何其他操作都将导致hbase崩溃。



不管是顺序还是随机读取，它们的效率都非常慢，从拥有10个列簇的表中读取数据的速度很快，但是随着列数的增加，速度也越来越低。

HBase建立在google的bigtable规范之上。Google的专家认为，列族的使用数量有限，一般来说不能超过几百个，这个数字确实可以达到500，但是性能却会大幅下降。在500多个列簇的HBase中，可以创建表，但是不能使用表[7,8]。如果想要建立一个拥有1000个列簇的表，那么这个请求将被拒绝。

# 改进

## 使用HBase作为MapReduce程序的数据源

在这个测试中，我们分析了hbase与mapreduce算法协同工作的方式。现在这个测试正在尝试模拟真实的情况，通常这种情况是基于非常大的日志文件。测试表结构与第一次测试相同，到目前为止没有任何变化。

程序要求如下：给定一个表用来存储网站上的每个用户访问的每个网页的信息，计算每个用户访问该网站的次数。该表由行ID和包含单个列的列簇组成。假定每个用户的访问数据被保存为一下格式:

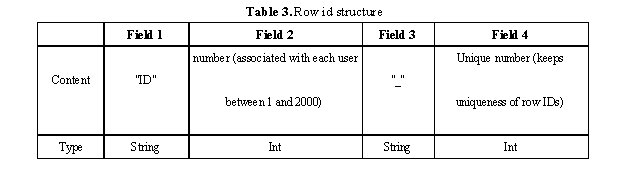
ID+user\_id+\_+timestamp组成行键。

列数据为访问页面的RUL地址。

使用10.000,100.000，300.000，500.000，700.000和1.000.000行的表来评估HBase + MapReduce的性能。对于每种情况执行以下步骤：

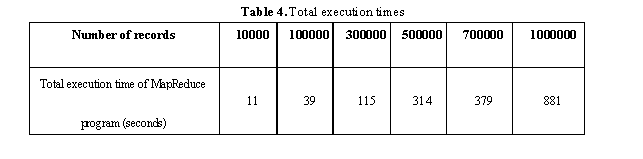
第一步：创建一个名为test的表，该表只包含一个名为colana1的列族，该列族中只包含一列，然后将随机生成的1000字节数据字符串按照行ID（依次）插入。

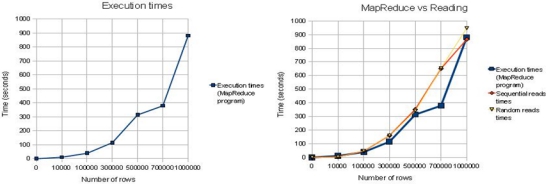
表3列出了每条记录的行号结构。



第二步：启动实现了MapReduce算法的脚本，它将对表中的所有数据进行扫描，然后将当前记录的行id传送给map()函数，该函数将删除主键中的“\_”并保证timestamp不重复。然后将数据处理成类似于（userid，1）的格式传送给Reduce()函数，Reduce()函数接受并处理收集到的键值对，以便将结果插入记录结果的result表中。

该脚本给出了下表中列出的结果。如图所示，执行时间随着行数的增加而增加，但并没有意外得增加太多。处理来自表中的100万条数据所花费的时间约为14分钟。Map步骤的平均执行时间约为94ms，Reduce步骤的平均执行时间约为32ms。





比较从这个测试（MapReduce程序执行）获得的结果和第一个测试（从数据库读取数据）的结果，可以发现，虽然在MapReduce程序中执行的操作比在HBase性能测试程序中执行的操作要多，即在一个有很多记录的表上（运行相同的数据提取操作加上它们在hbase的result表中的处理和插入操作），MapReduce程序执行时间较短。

这个结果是符合预期且合理的，原因如下：

* 在MapReduce中，记录不是通过脚本（顺序或随机）从数据库中提取的，而是通过HBase进行表扫描提取的，这个操作是HBase本身支持的，因此它可以更快的提取记录。
* Map()和Reduce()执行速度非常快，所以执行时间比较短。

# 结论

通过PL/SQL测试安装在redhat操作系统上的oracle服务器表明，对于非频繁的小数据集的查询oracle是可行的，但是在性能上和安装在centos 4.8的hadoop是没有办法进行比拟的。

HBase和面向列的其他非关系型数据库的比较，虽然它们在实施方面有很多不同，但是它们有着相同的目标：提供解决大数据集的存储和查询问题的方案。可能出现的问题表明，即使两者相差很大，两者之间的比较也是值得的。

正如前文所述，HBase是一个面向列的分布式数据库系统。HBase是Hadoop的延续，提供基于HDFS文件系统的数据存储和随机查询能力。从它构建开始，便遵循了以下一些原则：大量的行（数十亿的大小），大量的列（数百万的列）以及水平扩展能力。

表的结构反映了物理组织数据的方式，这个特性为数据的序列化，存储和提取提供了良好的支持。但是真正的问题在于应用程序和用户，他们需要学会以适当的方式管理存储空间。

从严格意义上来说，关系数据库是以遵循codd的12条规则为前提而定义的。一般的关系型数据库有一个固定的结构，具有ACID属性和针对于行列的强大的SQL引擎，特别是强一致性，参照完整性，以及通过SQL语言将任务从物理结构和复杂查询中抽象出来。我们可以很方便得创建二级索引，组合使用内部和外部链接，在行列上使用聚合函数等功能。

对于中小型应用来说，关系型数据库特别是以mysql为代表的自由对象关系型数据库有时是不可替代的东西，因为它们提供了简单性，灵活性且使用成熟度很高。但是当我们想要扩展到更大的数据量的时候，关系型数据库就变得不可使用了，因为它们的性能和可扩展性急剧下降。扩展信息实际上需要打破所有codd关系数据库规则，因此，关系型数据库的优势不在于处理非常大的数据集。

尽管hadoop似乎是世界上唯一的解决方案，当然指的是拥有大量数据的信息时代，在用户与软件交互的过程中，它的未来目标也会更加清晰。关系型数据库功能的易用性和使用范围很广很大程度上依赖于结构化查询语言。通过XQuery可以将Hadoop查询转化为SQL查询，它可以处理从XML文件中提取数据。通过这种方式，我们可以使用面对对象的程序和XQuery一起来构建基于XML的应用程序，Hadoop的另一个优点是不会在内存中存储任何空记录。

使用Hadoop和HBase平台，我们可以发现并行处理的优势，比如减少执行总时间。但是基于HDFS也有一些问题，比如数据需要被复制，所以总会占用相当一部分额外的空间。

HBase的另一个优点是可以随时添加或者删除一个或多个列族。为了添加或删除该列族，需要先禁用该表，这会导致在重新启用该表之前该表不可访问。

缺点是HBase不支持表之间的链接。但是这不是一个很严重的问题，因为可以将所有的数据都放在一张表中，并且可以很容易的获取到这些数据，这就消除了表连接的需求。

无论表中的数据数量有多少，从HBase中获取数据被证明都是相当快的，此外，HBase还提供了表扫描的功能，HBase+MapReduce被证明比顺序/随机读取更加快速，因为表扫描的操作是HBase原生支持的。HBase的另一个好处是可以搭配Zookeeper进行使用，Zookeeper可以代替主节点检查集群中机器的可使用性，从而为主节点分担压力，它同时也负责与客户端进行交互并返回交互结果。

参考文献

[1] R. Jones, Anti-RDBMS: A list of distributed key-value stores, http://www.metabrew.com/article/anti-rdbms-a-list-of-distributed-key-value-stores/.

[2] J. Sobel, Needle in a Haystack: Efficient Storage of Billions of Photos, http://perspectives.mvdirona.com/2008/06/30/FacebookNeedleInAHaystackEfficientStorageOfBill ionsOfPhotos.aspx.

[3] F. Chang, J. Dean, S. Et al, Bigtable: A Distributed Storage System for Structured Data, OSDI 2006.

[4] R. Rawson, HBase committer, HBase, http://docs.thinkfree.com/docs/view.php?dsn=858186

[5] HBase, www.apache.org/hadoop/HBase/HBaseArchitecture

[6] A. Khetrapal, V. Ganesh, HBase and Hypertable for large scale distributed storage, systems: A Performance evaluation for Open Source BigTable Implementations, http://www.ankurkhetrapal.com/downloads/HypertableHBaseEval2.pdf

[7] A. Rao, S. Zang, HBase-0.20.0 Performance Evaluation, http://cloudepr.blogspot.com/2009\_08\_01\_archive.html

[8] K. Dana, Hadoop HBase Performance Evaluation, http://www.cs.duke.edu/~kcd/hadoop/kcd-hadoop-report.pdf

[9] J. Graz, J. D. Crzans, HBase goes Realtime, The HBase presentation at Hadoop Summit 2009.

[10] HBase-0.20.2 Documentation, [http://hadoop.apache.org/hbase/docs/r0.20.2.](http://hadoop.apache.org/hbase/docs/r0.20.2)

[11] K. Loney, "Database 10g - The Complete Reference", Ed Mc Graw Hill, 2004.

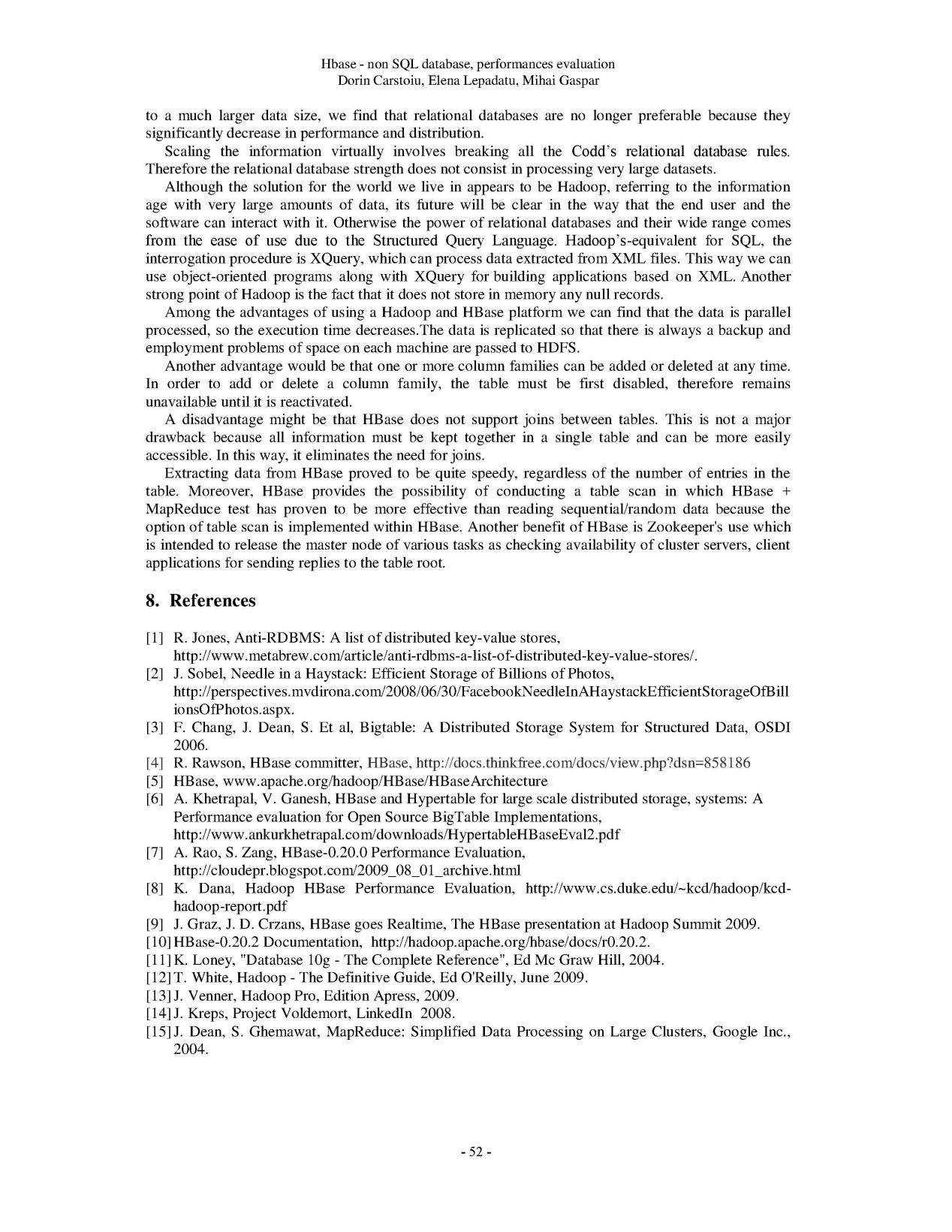
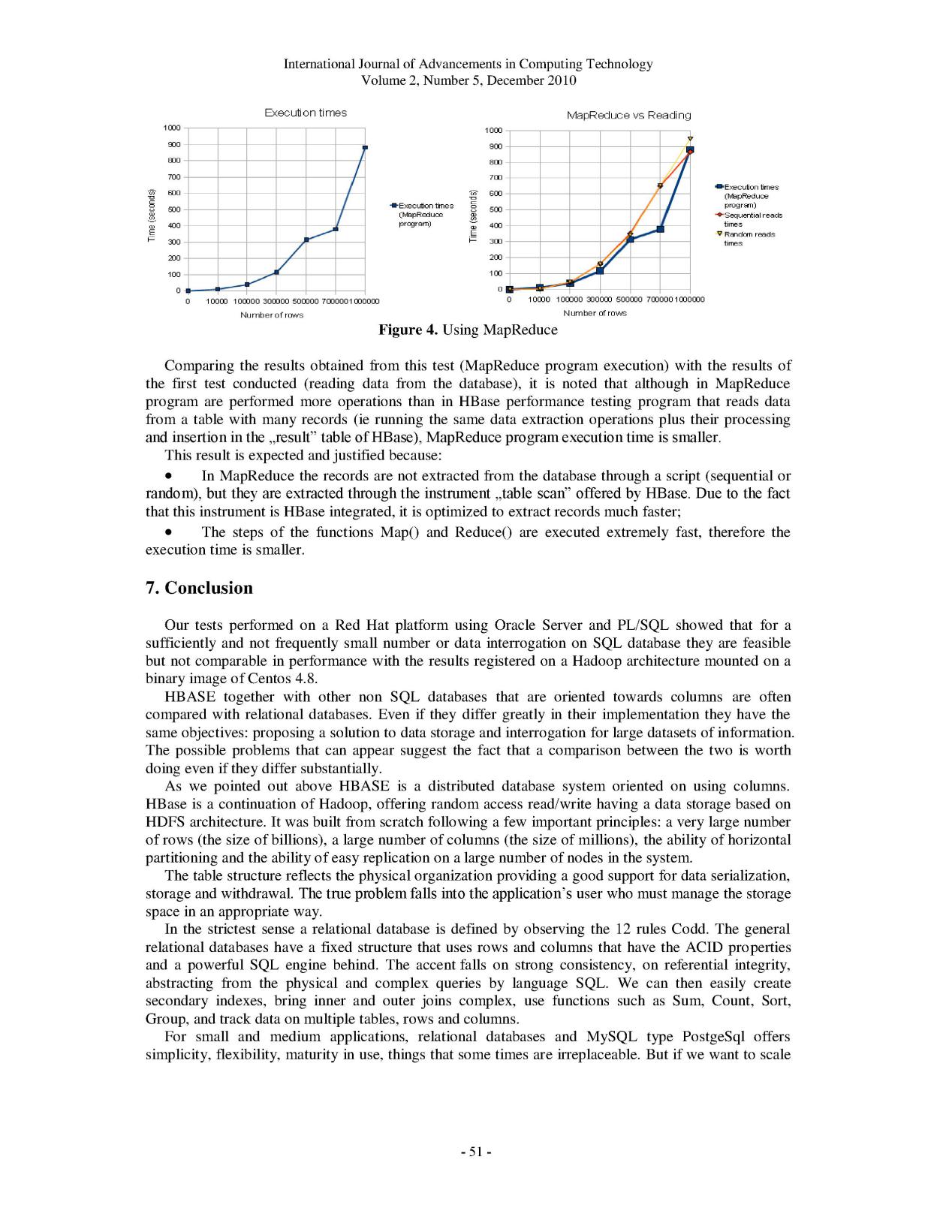
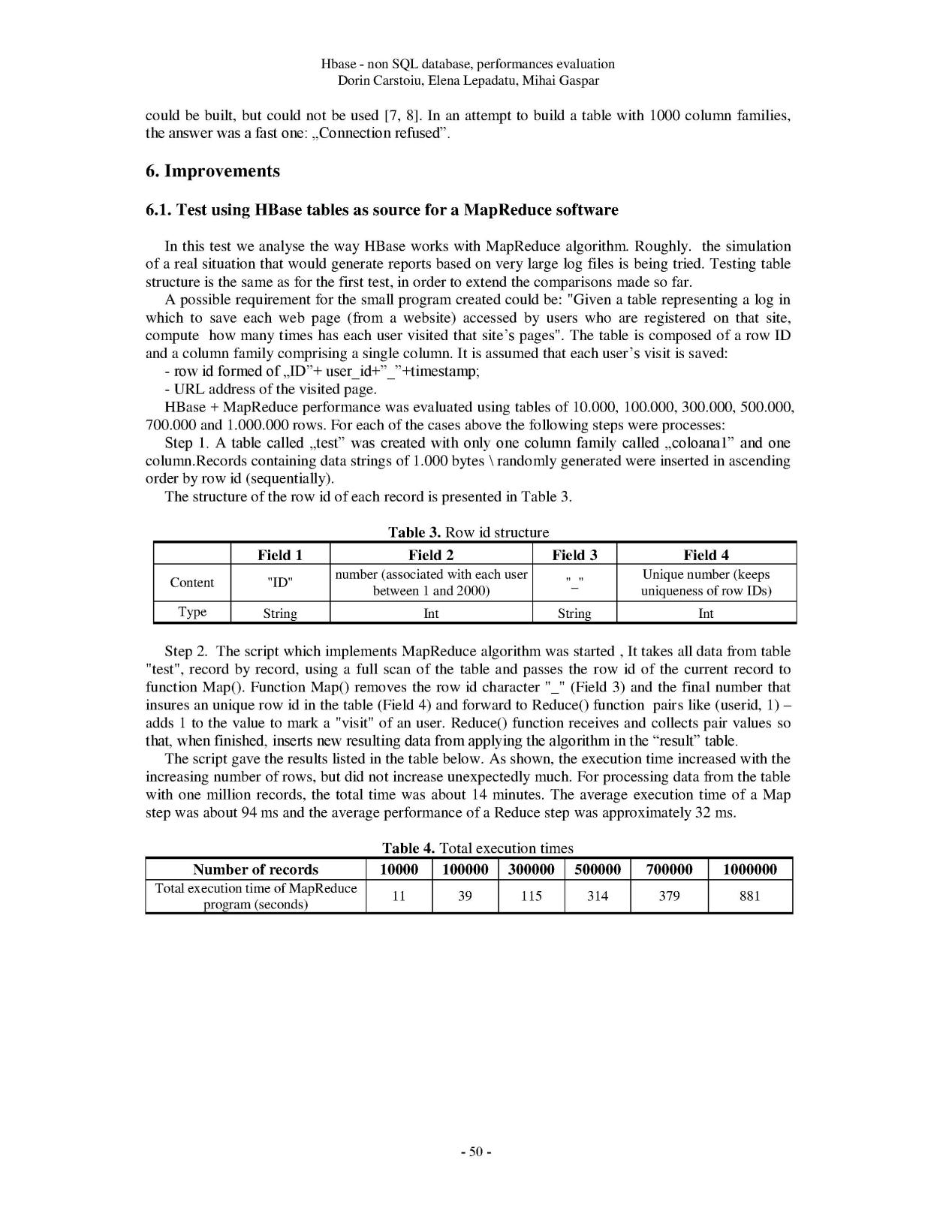
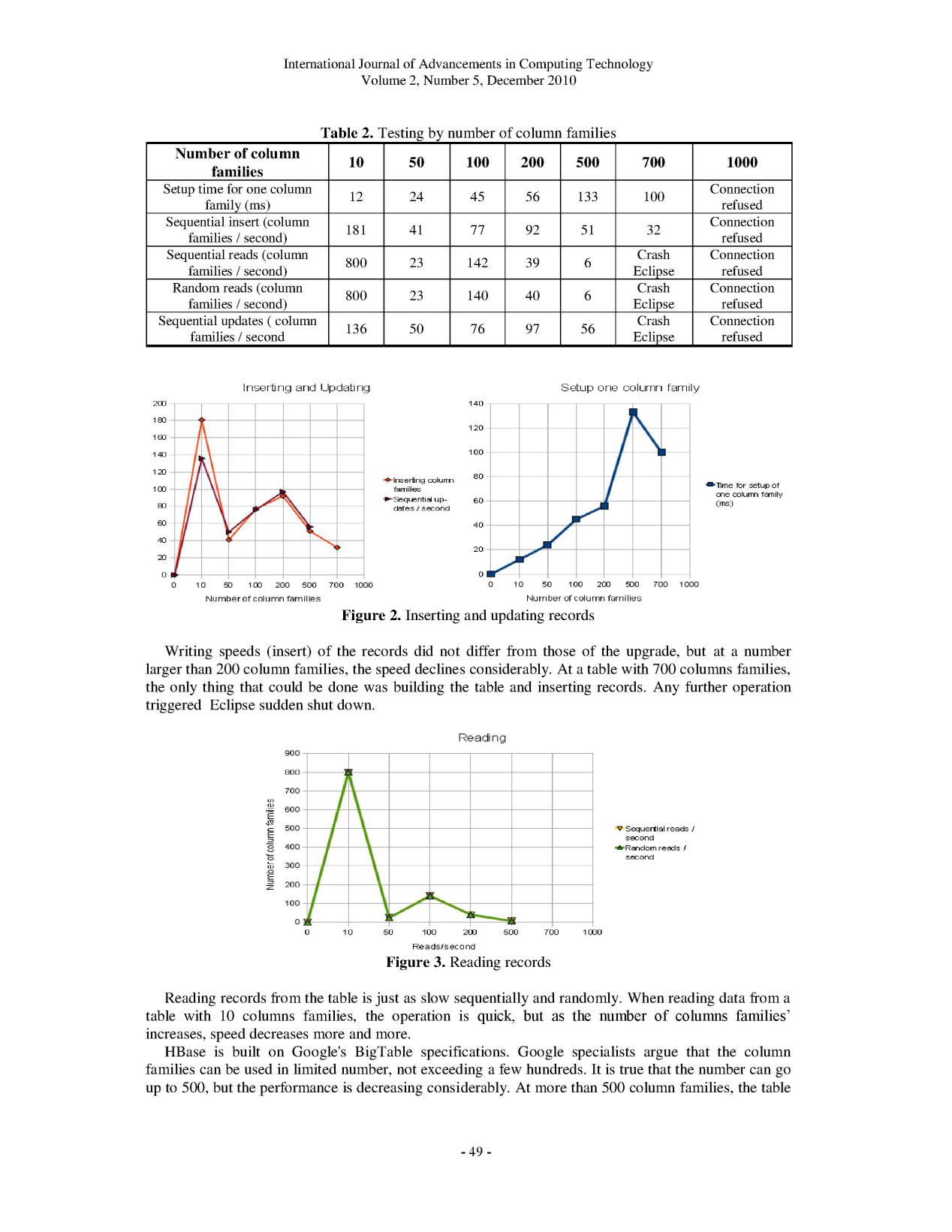
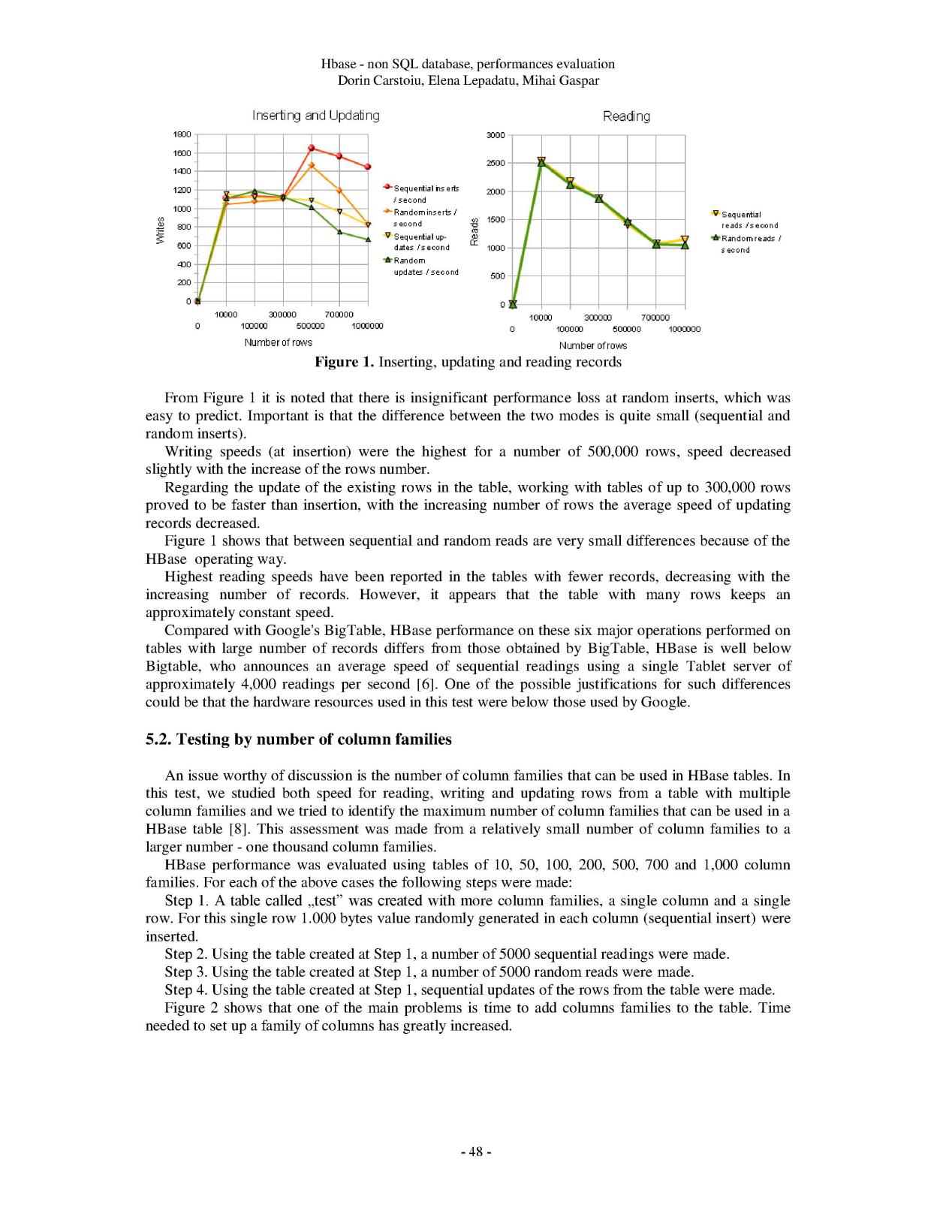
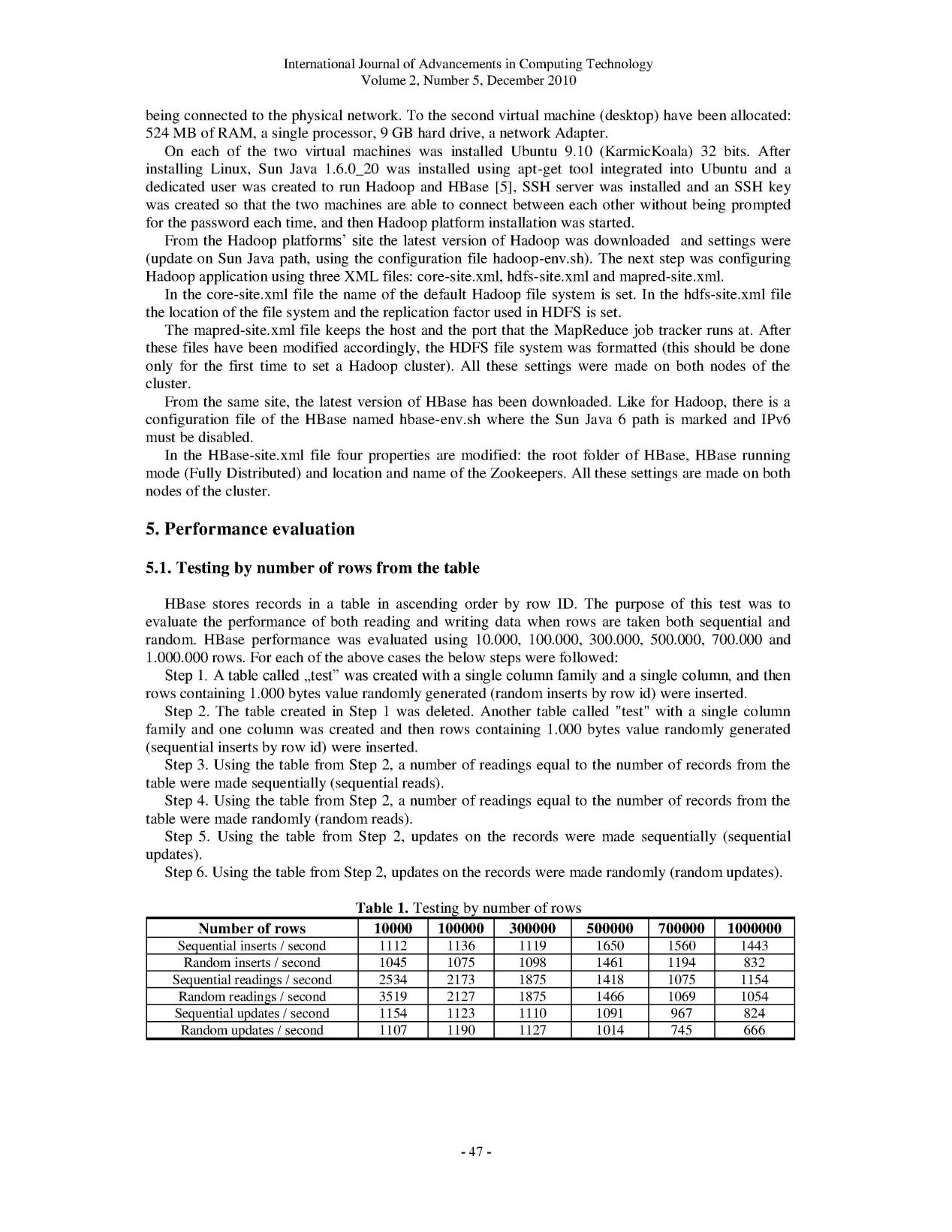
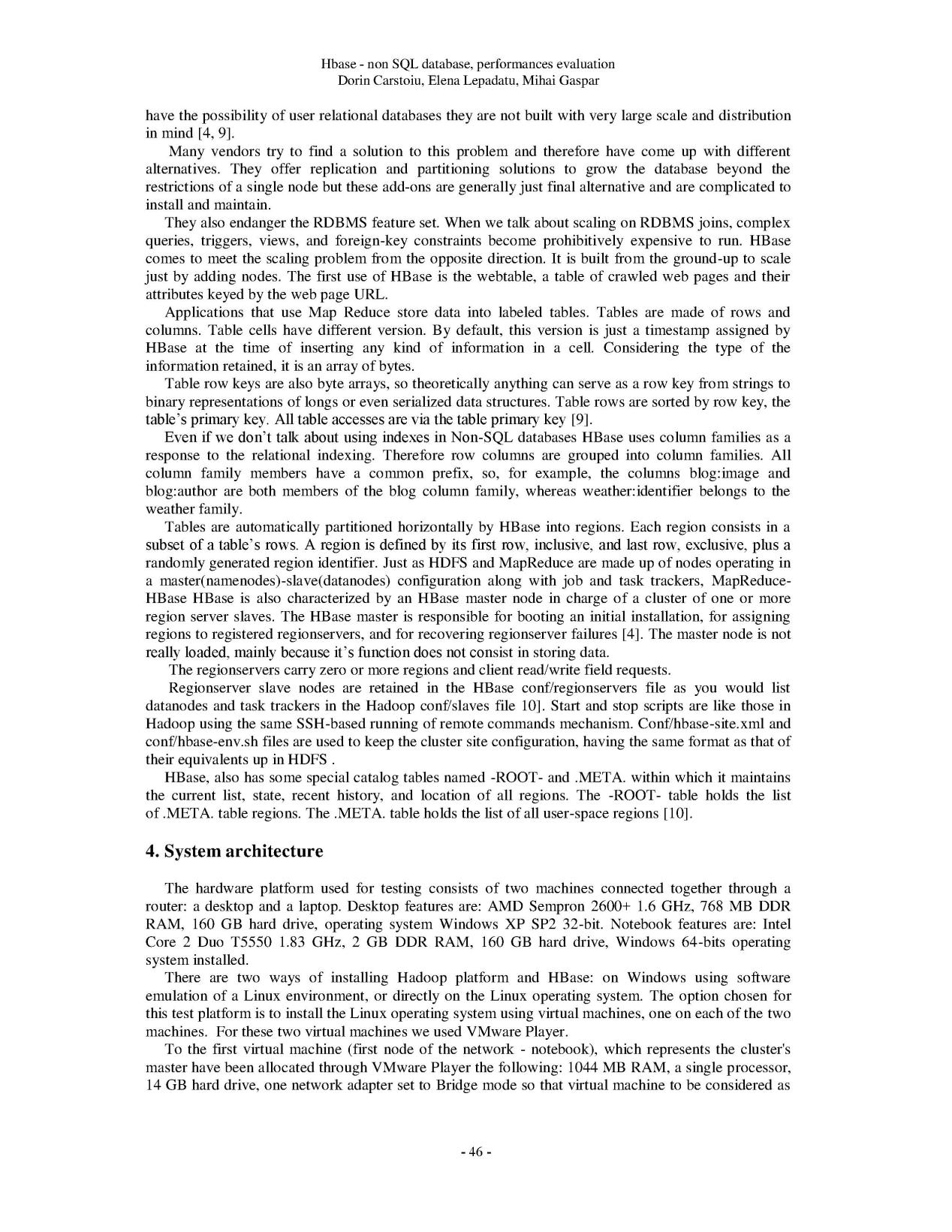
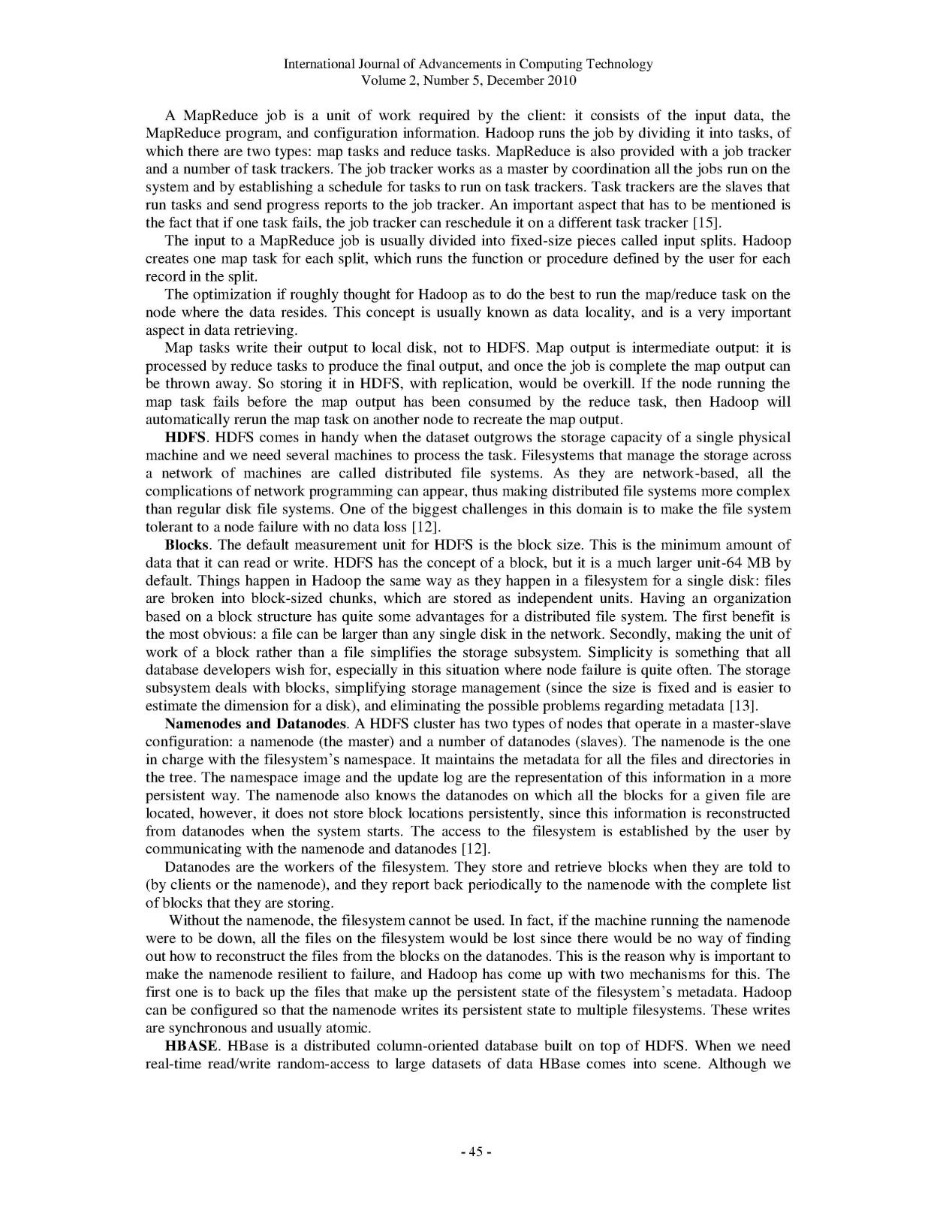
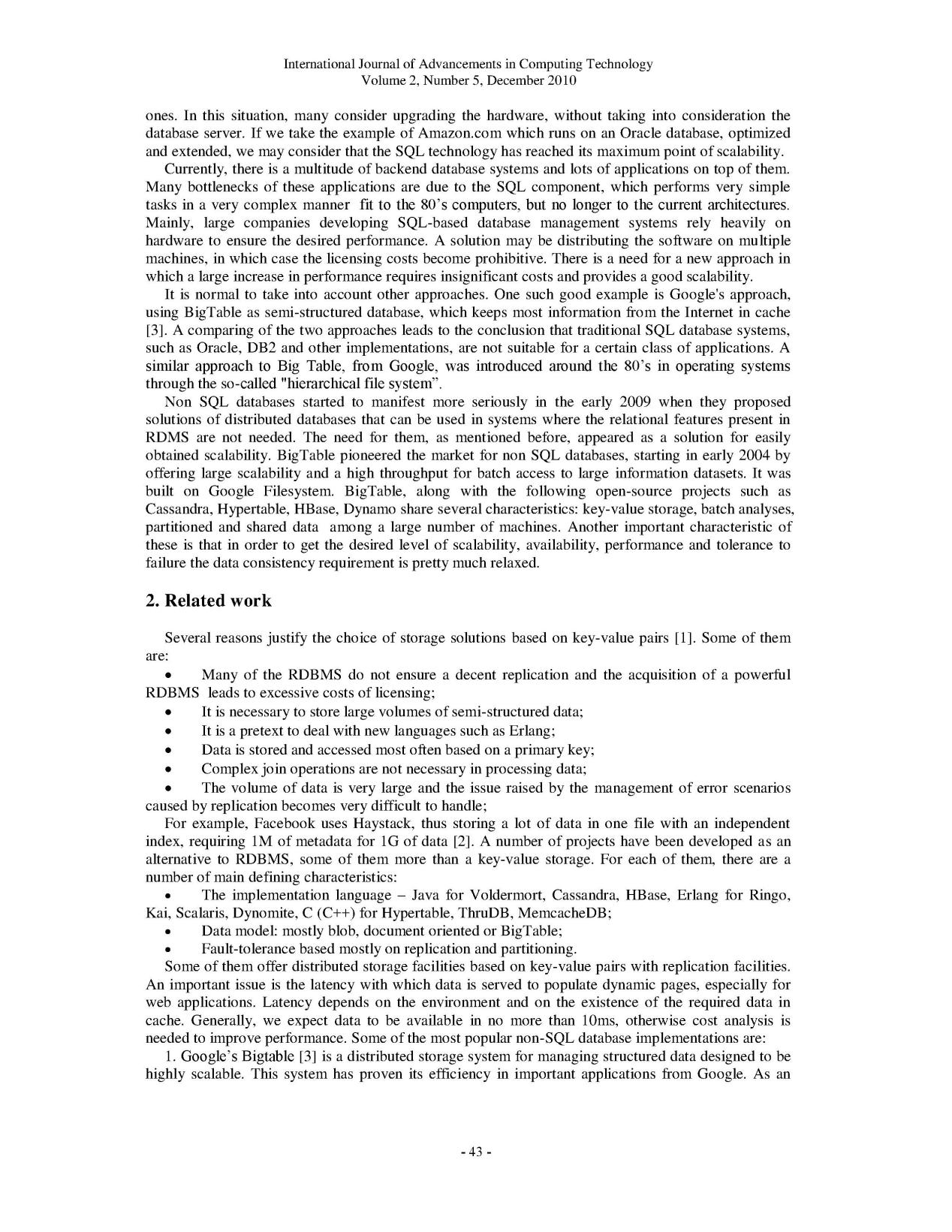
[12] T. White, Hadoop - The Definitive Guide, Ed O'Reilly, June 2009.

[13] J. Venner, Hadoop Pro, Edition Apress, 2009.

[14] J. Kreps, Project Voldemort, LinkedIn 2008.

[15] J. Dean, S. Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, Google Inc., 2004.

**附英文原文：**

****