Table of Contents

1. Introduction 2

1.1 Background: 2

1.2 Aims and Objectives of the project 3

1.3 Structure of the Report 3

2.   Literature Review 4

2.1 Model Driven Engineering 4

2.2 EMF 5

2.2.1 Ecore metamodel 5

2.2.2 Modeling in EMF 5

2.2.3 Modelling persistence 6

2.3 NoSQL database 8

2.3.1 Introduction 8

2.3.2 Why NoSQL appear, become popular & reason 8

2.3.3 CAP??????????? Write here or where??? 9

2.3.4 Current NoSQL database (type) 9

2.3.5 Model persistence of graph database (why particular graph db is suitable for model persistence)? 9

2.3.6 Blueprint API 9

2.3.7 What are the NoSQL problems & different db are used in different situation, what are the resons ? how to solve it. 10

RDF ??????? 10

3. Methodology 10

3.1 Get the baseline of the NoSQL databases 10

3.1.1 why need baseline 10

3.1.2 chose the db: orientDB, sesame, postgresql and why 10

3.2 identify the bottlenecks 10

3.2.1 for each db, what is the problems, or bottlenecks 10

3.2.2 blueprints 10

3.2 Do some relative research and find some way to improve performance 11

3.3 implement in small code and see if it increase performance 11

3.4 use the prototype into the blueprint 11

3.5 compare with the base line and see the differences. 11

4. Implementation 11

4.1 Get the base line 11

4.2 Identify bottlenecks 13

About blueprints 13

4.3 Provide relative solutions (Prototype) 13

4.4 New Result: 15

# 1. Introduction

**//**noSQL→ blueprints (same time)

A database is a collection of data that is organized in such a manner that allows data to be stored and traversed in an easier way. Complexity of traversing and querying starts to increase with the size of the databases. Currently, there are numerous database technologies present and all of them tend to have their benefits and drawbacks. Recently few databases – namely: Orient DB, Sesame DB, Neo4J came into the market that offers the storage and querying of data in a different manner than the traditional way of stacking up data. A few of them uses graph technology to store and establish relationship between them. However, in terms of performances, some of the databases that implement this technology are not up to the mark yet. There tend to be huge room for improvement of these databases.

## 1.1 Background:

Problems:

Several cloud-based data storage systems (often called NoSQL) are emerging that are quite different to traditional relational data storage. Each kind of store is designed with a specific application and a demanding workload in mind, that relational databases cannot easily handle. Some data stores are also suitable for storing large models of software. However, each kind of data store has a unique performance profile that means models must be stored using a specific format for that data store to be preferment.

### Why NoSQL appear, become popular & reason

Appear reason:

 \* object-relational mapping can be complex or costly

 \* relational db do not play well with dynamically structured data and often-varying schemas

\* overhead of sql parsing and full-blown query engines maybe significant for simple access patterns(primary key access, BLOB storage etc.)

\* scaling to many servers with the ACID guarantees provided by relational db is hard

Why NoSQL is popular these days

·      Avoidance of Unneeded Complexity

·      High Throughput

·      Horizontal Scalability and Running on Commodity Hardware

·      Avoidance of Expensive Object-Relational Mapping

·      Complexity and Cost of Setting up Database Clusters

·      Compromising Reliability for Better Performance

<http://www.techrepublic.com/blog/10-things/10-things-you-should-know-about-nosql-databases/>

### 2.3.3 CAP??????????? Write here or where???

[move to intro]

## 1.2 Aims and Objectives of the project

The aim of the project is to increase the performance of current NoSQL database that could save the large-scale model and keep model persistence. This is to be done by discovering their bottlenecks and investigating the most suitable form for persisting large models in NoSQL data stores.

Upon accomplishing that task, the objective is to look deeply into one or two of the database technologies and scrutinize its processing power to improve and harness its performance. In short, comparing two or more persistence formats for a given kind of data store.

## 1.3 Structure of the Report

The report will start off with a Literature review of the model driven engineering that uses software engineering principles. It will be looked into with greater detail about how one of the specific database technology works – NoSQL. After that, previous works done by other scientists will be discussed: their solutions’, benefits and limitations.

Next section onwards should start off with focusing on the developers work. The requirement and analysis would be discussed, followed by detailed description of implementation and evaluation and finishing off with a conclusion.

# 2.   Literature Review

## 2.1 Model Driven Engineering

[find one journal paper based on this topic and rewrite]

[why do we need models and what are the models]

[model transformation]

[gives the person an understanding so computer scientists can understand]

Model Driven Engineering is an approach to software development that elevates ‘modles’ to first class artefacts of the software engineering process.

[*Hawk: Towards a Scalable Model Indexing Architecture*]

??????????????? What should I write here? Just the explanation?

What is mde, intro journal papers

Model transformal. Models tool.

**2.2 Scalability**

[one big section about scalability]

[models can be big]

[models that are 700mb, describe any one of them]

[massive models]

Half-one page

## 2.3 EMF

Eclipse Modelling Framework is often in collaboration with various modelling standards created by the Object Management Group (OMG). OMG is an international, non profitable consortium responsible for developing modelling standards for enterprise implementation using various technologies. The following section will describe these standards and their relationship with the EMF.

### 2.2.1 Ecore metamodel

The model used to represent models in EMF is called Ecore. Ecore is itself an EMF model, and thus is its own metamodel.

An essential part of EMF is the Ecore (Meta) Model or just Ecore. Ecore is a model that is used to represent the types of information which exist in instances of EMF models. Hence it can be said that Ecore is its own metamodel [11]. Ecore serves not only as an essential component for acquiring the model’s classes and interfaces but also as a tool for transformation purposes. This is accomplished mainly due to the mapping of the models structural characteristics to the Ecore classes. Error! Reference source not found. shows the full architecture of the Ecore metamodel in terms of elements, associations and hierarchy.

### 2.2.2 Modeling in EMF

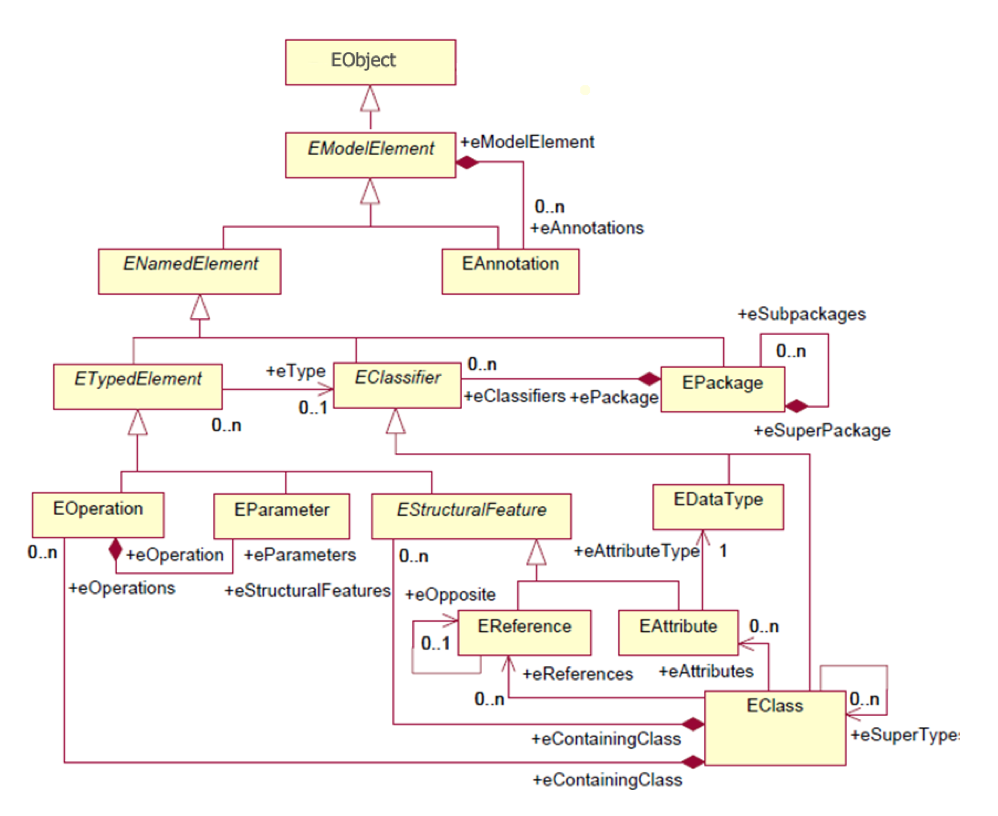
Generally four Ecore classes are needed to represent a model

1.     ECLass

2.     EAttribute

3.     EReference

4.     EDataType



Ecore matamodel architecture

### 2.2.3 Modelling persistence

EMF provides two database solutions for persisting instances of EMF models: Teneo and Connected Data Object (CDO). Both of these EMF Technology (EMFT) components persist models to a relational database (although, during the writing of this thesis it was announced that CDO’s latest upgrade would persist models in the non-relational MongoDB databases, as well). These database mechanisms were originally created to handle the ever growing size of models and provide a dynamic solution to problems related to the lack of loading efficiency and modularity that the default XMI persistency solution suffered from [Dimitrios S. Kolovos, Paige F. Richard, and Fiona A.C. Polack, "The Grand Challenge of Scalability for Model Driven Engineering," 2009]. The following sections describe the two existing technologies and emphasize on the characteristics of these EMFT components in terms of scalability and usability.

* **Teneo:**

Teneo’s persistency solution is achieved via Hibernate and EclipseLink [1]. As a Relational Database solution, Teneo provides domain specific mappings and the creation of database schemas based on the instance of EMF models as mention in section 2.4.2. In general, objects can be managed with the use of Hibernate Query Language (HQL) and EJB-QL queries [(2010) Eclipse website. [Online].

http://wiki.eclipse.org/images/7/7e/EclipseProjectEvaluations.pdf]. Furthermore, Teneo supports another type of mapping called Entity-Attribute- Value (EAV). In contrast to the default domain-specific mapping, EAV mapping stores the information in only 2 tables: one for Objects and one for Values. Consequently, Teneo is more suited for the development of RCP and web service standard applications were the database schema remains the same even when the instance of an EAV mapped model changes. Nevertheless, Teneo’s model persistence is not restricted to one of the two choices, either a standard domain mapping or and EAV mapping but it can also support a combination of both. Essentially, a section of the model can be mapped with the use of the EAV mapping mechanism and another section with the default domain mapping. This practically means that the database schema requires the creation of tables suitable for handling both approaches simultaneously.

**[1]** Eclipse. [Online]. http://wiki.eclipse.org/Teneo

* **CDO:**

<http://wiki.eclipse.org/images/7/7e/EclipseProjectEvaluations.pdf>

CDO is considered a distributed solution that supports object relational mapping between EMF models and relational databases. In contrast to Teneo, CDO commits the changes to the relational database internally when the Resource is saved. Additionally, this method can be adjusted so that it notifies other possible clients by invalidating their EMF Applications so that they re-load the model from the repository. However, as mentioned in [Martin Flügge and Eike Stepper, "Scale, Share and Store your Models with CDO," Eclipse Summit Europe, November 2010.] this approach is not suitable for multi-client environment. Nevertheless, CDO handles very well large size of models and provide features like: lazy containment loading, adaptive prefetching and partial collection loading that increase its scalability [(2010) Eclipse website. [Online].

http://wiki.eclipse.org/images/7/7e/EclipseProjectEvaluations.pdf]. Furthermore, an important characteristic of CDO’s functionality is memory management. Objects which are no longer in use are made available for reclaim by the garbage collector thus releasing occupied memory. Consequently, CDO has become a widely used tool for developing significant commercial software applications.

* **XMI**

XML Metadata Interchange is no more than a format, based on the MOF standard in which a model corresponds to an optimized XML document called XMI Schema. [OMG: XML Metadata Interchange Specification, version 1.2, OMG document 02-01-01]

XMI schemas use certain rules for structuring the hierarchy of a model. They maintain the names and elements of the model and provide a new containment hierarchy that starts from the packages up to the simplest elements of the classes. This hierarchy is named XMI Serialization or Ecore XMI if we are referring to an Ecore model. One of the most important aspects of XMI Serialization is that it provides an infrastructure for exchanging UML models created by different tools. This dynamic XMI-based transformation makes the exchange of two different models - as far as they follow the Meta Object Facility standards - possible. [Jernej Kovse and Theo Harder, "Generic XMI-Based UML Model Transformations," in 8th International Conference on Object-Oriented Information Systems, Montpellier, France, 2002.]

Furthermore, this feature in combination with the persistence framework is the core of EMF. The next section describes how XMI Serialization mechanism is used to manage object persistence.

Importance of Model persistence:

An important mechanism that EMF provides is model persistence. As mentioned above, Ecore models are stored using XMI Serialization as the default standard. However, EMF includes the feature of loading, storing and persisting objects that derive from various models, not only Ecore. In addition, the instances of objects can be persisted in a specified form and in an adjustable encoding configured by the developer.

## 2.3 NoSQL database

### 2.3.1 Introduction

NoSQL, or 'Not Only SQL', represents the new class of data management technologies designed to meet the increasing volume, velocity, and variety of data that organizations are storing, processing, and analyzing.

[http://www.mongodb.com/nosql-explained]

NoSQL databases are becoming an increasingly important part of the database landscape, and when used appropriately, can offer real benefits. However, enterprises should proceed with caution with full awareness of the legitimate limitations and issues that are associated with these databases.

In the past few years, the ”one size ﬁts all“-thinking concerning datastores has been questioned by both science and web aﬃne companies, which has lead to the emergence of a great variety of alternative databases. The movement as well as the new datastores are commonly subsumed under the term NoSQL,

“used to describe the increasing usage of non-relational databases among Web developers”

[*Obasanjo, Dare: Building scalable databases: Denormalization, the NoSQL movement and Digg. September 2009. – Blog post of 2009-09-10*]

### 2.3.4 Current NoSQL database (type)

Key-value –

·      Introduction

·      db

·      used in which situation

**Family**: Amazon's Dynamo paper and Distributed HashTables.

**Data model**: A global collection of KV pairs.

**Example**: Membase, Riak

**Good** at: Handles size well. Processing a constant stream of small reads and writes. Fast. Programmer friendly.

·      not suitable for what - ?

Document based db: Mongo DB

·      Introduction

A document-oriented database is a computer program designed for storing, retrieving, and managing document-oriented information, also known as semi-structured data. Document-oriented databases are one of the main categories of NoSQL databases and the popularity of the term "document-oriented database" (or "document store") has grown[1] with the use of the term NoSQL itself. In contrast to relational databases and their notions of "Relations" (or "Tables"), these systems are designed around an abstract notion of a "Document".

·      db: Mongo DB

·      used in which situation:

To operate over a wide variety of access patterns and data types then look at a Document database, they generally are flexible and perform well.

**Family**: Inspired by Lotus Notes.

**Data** **model**: Collections of documents, which contain key-value collections.

**Example**: CouchDB, MongoDB

**Good** at: Natural data modeling. Programmer friendly. Rapid development. Web friendly, CRUD.

·      not suitable for what

Wide Column Store/Column-Family

·      Introduction

·      db

·      used in which situation

·      not suitable for what

Wide column stores, also called extensible record stores, store data in records with an ability to hold very large numbers of dynamic columns. Since the column names as well as the record keys are not fixed, and since a record can have billions of columns, wide column stores can be seen as two-dimensional key-value stores.

Wide column stores share the chracteristic of being schema-free with document stores, however the implementation is very different.

Cassandra

HBase

Accumulo

Graph based database

·      Introduction

·      db

·      used in which situation

·      not suitable for what

**Family**: Euler and graph theory.

**Data** **model**: Nodes & relationships, both which can hold key-value pairs

**Example**: AllegroGraph, InfoGrid, Neo4j

**Good** at: Rock complicated graph problems. Fast.

Focus on graph and triple, but talk about others too briefly.

Why they are suitable

### 2.3.5 Model persistence of graph database (why particular graph db is suitable for model persistence)?

### 2.3.6 Blueprint API

graph abstraction of different types of databases

because graph databases are suitable for models

### 2.3.7 What are the NoSQL problems & different db are used in different situation, what are the resons ? how to solve it.

[not here, need more intro about methodology and problem should come as a conclusion]

[you haven’t explained the problem and need to have concrete system]

[problem – blueprints is a great drive, some cases it uses too much time, too much memory, in some cases emf is not suitable]

[no guidance that tells me optimize for different kinds of servers]

🡪MDE (model persistence)

*NoSQL advocate Ben Scoﬁeld responds to some of the criticism mentioned above which he perceives to be misconceived. He does so by expressing incisive arguments from the NoSQL debate responding to them*

·      “NoSQL is just about scalability and/or performance.”

·      “NoSQL is just document databases, or key-value stores, or . . . ”

·      “I can do NoSQL just as well in a relational database.”

·      “NoSQL is a wholesale rejection of relational databases.”

no common interface/query language

Based on application requirement, the type of database technology that is required is discussed below:

If the application NEEDS 🡪

* **complex transactions**   
  data cannot be lost, or if a simple transaction programming model is needed, then Relational or Grid database is a good choice.

Example: an inventory system that might want full ACID transactions. A user could be really upset when they buy a product and later it was shown its not available in stock. Some might be happy with a substitute but most users would not be, they basically want the desired item.

* **to scale** then NoSQL or SQL both can work. Systems that support scaling, partitioning, live addition and removal of machines, load balancing, automatic sharding and rebalancing, and fault tolerance is a good choice.
* to **always** be able to **write** to a database because it is needed for high availability. Bigtable Clones which feature eventual consistency is a good place to go too as well.
* Numerous small **continuous reads and writes**, that may be volatile, then Document or Key-value or databases offering fast in-memory access is a good choice. Also SSD could be considered as well.
* to implement **social network operations** then a Graph database or second, a database like Riak that supports relationships is a good alternative. An in- memory relational database with simple SQL joins might suffice for small data sets. Redis' set and list operations could work too.

## RDF ???????

## Resource Description Framework. The RDF data model[reference] is similar to classic conceptual modeling approaches such as entity–relationship or class diagrams, as it is based upon the idea of making statements about resources (in particular web resources) in the form of subject-predicate-object expressions. These expressions are known as triples in RDF terminology

## [http://www.w3.org/TR/PR-rdf-syntax/ "Resource Description Framework (RDF) Model and Syntax Specification"]

## Triple stores

A triplestore is a purpose-built database for the storage and retrieval of triples,

Much like a relational database, one stores information in a triplestore and retrieves it via a query language. Unlike a relational database, a triplestore is optimized for the storage and retrieval of triples. In addition to queries, triples can usually be imported/exported using Resource Description Framework (RDF) and other formats

*[TripleStore, Jack Rusher, Semantic Web Advanced Development for Europe (SWAD-Europe), Workshop on Semantic Web Storage and Retrieval - Position Papers]*

# 3. Methodology

## 3.1 Get the baseline of the NoSQL databases

### 3.1.1 why need baseline

Whenever a performance issue of any product is in question, the best way to answer is to provide comparison with the competitive products. To do this, a baseline is of paramount of importance. Since having a baseline means it allows introduction of the product of its potential and its capabilities are vivid.

This baseline is developed to produce a measure of comparison to see how much enhancement has been made compared to the previous works.

### 3.1.2 chose the db: orientDB, sesame, postgresql and why

Numerous databases were looked into. More focus and priority were given to the databases: PostgreSQL, OrientDb and sesame DB. These databases were chosen since they are emerging in the current market with good adoption rate.

Postgresql:

It has greater scalability features and also involves JSON support. Developers and IT managers find these features lucrative for their business and company too. Since this database technology is something consumers are after and if one could improve its performance, it would benefit a huge number of people. In a survey conducted by EnterpriseDB, it was found that moving to Postgres Plus Advanced server from an Oracle one yielded $1,706,673 over a 3-year cumulative benefit

[link: http://www.slideshare.net/insideHPC/postgres-survey-podcast] via moving on to Enterprise DB PPAS on HP integrity BL890c server blades. These gives more reasons to aim this database and focus research on its improvement.

## 3.2 identify the bottlenecks

### 3.2.1 for each db, what is the problems, or bottlenecks

PostgreSQL:

A development of software depends on numerous entities. If the scale of software is not very small, it is easy to coagulate the performance via the creation of bottlenecks. As such, to improve the performance of these software systems, the discovery of bottlenecks is of paramount importance.

Scrutinizing Postgresql, it was discovered that the additions of Nodes and References are the biggest bottlenecks of the database. They are hinging the performance of the overall database. Whereas all the other processes were taking average time, the specific part of additions of references and nodes were taking the bulk of the time.

These bottlenecks were pin-pointed down using a profiler – Jprofiler, and it was seen most of the time the program has been running; the majority of the run-time is being spent on the addition of nodes and references.

### 3.2.2 blueprints

3.2.3 memory leaks, which parts are slow, bottlenecks

## 3.2 Do some relevant research and find some way to improve performance

## 3.3 implement in small code and see if it increase performance

Steps to implement solutions were done in small code segments. In the beginning, methodologies and strategies were placed out. Specific sections of the project and the source code were looked at. The same mechanisms that were being used within the Postgresql project were ported into a new project to test certain features of it.

Looking at loops is a common place to start off with, loops were used to carry out a routine task. However, in some cases, the loops have not being sufficiently adequate in terms of efficiency. Therefore to research the fact, should there be any performance gain, via tweaking loops were done via implementing small code in a different project.

Research was done in Eclipse Keplar, where a new project was created and specific features tested with new tweaks and enhancements. The results have been recorded over a series of 10 runs and the average time taken from these results were carried forward as the baseline of the small code base.

## 3.4 use the prototype into the blueprint

## Once the small code segments started yielding results and proved to be promising in terms of performance using different techniques. All those small codes were integrated and put forward to develop a prototype that would be used later on in the final deliverable.

## This prototype was built in accordance to the blueprints driver that is available. It is well synchronised with the benchmark project and is test ran with four different data sets.

## 3.5 compare with the base line and see the differences.

## Initially, before the start of the development process, specific set of baselines were made and upon completion of the prototype, these baselines came to light. The prototypes were benchmarked for performance analysis, and baseline profiles for them are made respectively. The baselines were finalized upon recording all the data sets for a minimum of ten runs. The final versions of the baselines are an average of ten runs, which is deemed to be having an adequate level of integrity for the results.

## The previously generated baselines of the initial Blueprints were compared with the latest ones and the results were vivid. The conclusion drawn from these are clear and is also illustrated in the table below.

|  |  |  |  |
| --- | --- | --- | --- |
| Baseline |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## [3.3 – 3.5, should be strategy, make prototype, compare and blab la]

# 4. Implementation

# what I did to improved, what changes u made

# where did I looked for

# what prototype I made [all], useful. And how.

# How it helped us

# 5.Results

# Analyze the results

## 4.1 Get the base line

**Base Line:**

**Configuration options for benchmarks**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Config | OrientDB  Native API | OrientDB  Blueprints | Sesame  Native | Sesame  XMI to N3 | PostgreSQL |
| JVM | -Xmx8g | -Xmx8g | -Xmx8g | -Xmx8g | -Xmx8g |
| Database | Default | Default | Default | Default | Default |

OrientDB results:

Time = load + create nodes + add references + unload + commit

|  |  |  |
| --- | --- | --- |
| [ set 0 ] | Time | Memory |
| native: | 0, 7, 6, 0, 0  = 13 |  |
| blueprints: | 0, 3, 6, 0, 1  = 10 | 142MB |

|  |  |  |
| --- | --- | --- |
| [ set 1 ] | Time | Memory |
| native: | 1, 20, 15, 0, 0 = 36 |  |
| blueprints: | 1, 8, 18, 1, 3 = 31 | 282MB |

|  |  |  |
| --- | --- | --- |
| [ set 2 ] | Time | Memory |
| native: | 12, 310, 123, 4, 0 = 449 |  |
| blueprints: | 12, 69, 206, 8, 2 = 297 |  |

|  |  |  |
| --- | --- | --- |
| [ set 3 ] | Time | Memory |
| native: | 64, 836, 334, 11, 0 = 1245 |  |
| blueprints: | 65, 176, 520, 19, 2 = 782 |  |

|  |  |  |
| --- | --- | --- |
| [ set 4 ] | Time | Memory |
| native: | 118, 833, |  |
| blueprints: | 119, 153, 452, 20, 0 = 744 |  |

Sesame result:

Time = load + nodes + references + unload + commit

|  |  |  |
| --- | --- | --- |
| [ set 0 ] | Time | Memory |
| Native : | 0,5,0,0,0🡪 total:9 |  |

|  |  |  |
| --- | --- | --- |
| [ set 1 ] | Time | Memory |
| Native : | 1, 15, 2, 0, 4(commit)—> total: 25 |  |

|  |  |  |
| --- | --- | --- |
| [ set 2 ] | Time | Memory |
| Native : | 13, 198, 47, 5, 41(commit)—> total: 307 |  |

|  |  |  |
| --- | --- | --- |
| [ set 3 ] | Time | Memory |
| Native : | 66, 702, 182, 9, 125(commit)—>total: 1087 |  |

|  |  |  |
| --- | --- | --- |
| [ set 4 ] | Time | Memory |
| Native : | 105, 753, 172, 10, 139(commit)—>total: 1181 |  |

## 4.2 Identify bottlenecks

**PostgreSQL problems:**

The source code for PostgreSQL was downloaded off the Internet from the GitHub repository. The initial analysis of the PostgreSQL when ran with a set of four datasets, of varying data size limits, the following deductions were drawn:

[Data Set 1]: Completes execution completely and successfully.

[Data Set 2]: Execution cannot finish. Does not throw error.

[Data Set 3]: Execution terminated with ‘Out of Memory’ error.

[Data Set 4]: Execution terminated with ‘Out of Memory’ error.

Reason:

It was discovered that the parallel collector will throw an OutOfMemoryError if too much time is being spent in garbage collection: if more than 98% of the total time is spent in garbage collection and less than 2% of the heap is recovered, an OutOfMemoryError will be thrown.

This feature is helpful to debug and is designed to prevent applications from running for an extended period of time while making little or no progress because the heap is too small [Ref]. If necessary, this feature can be disabled by adding the option -XX:-UseGCOverheadLimit to the command line. –

[link: <http://www.oracle.com/technetwork/java/javase/gc-tuning-6-140523.html>]

See more at: http://www.jvmhost.com/articles/what-is-java-lang-outofmemoryerror-gc-overhead-limit-exceeded#sthash.9PhEjFsN.dpuf

Postgres doesn't use Random Access Memory. However, it basically depends on the Operating System’s file system page cache for the bulk of its caching, therefore monitoring the RAM usage on a system running PostgreSQL, it is being observed that many GBs of memory are in use by the OS buffers/cache whereas individual PostgreSQL backend processes takes up only few tens of Megabytes.

The ‘Out of memory’ message also means that for some reason the garbage collector is taking an excessive amount of time (by default 98% of all CPU time of the process) and recovers very little memory in each run (by default 2% of the heap).

This effectively means that the program has stopped making any progress and is busy running only the garbage collector all time. To prevent the application from soaking up all the CPU time without getting anything done, the JVM throws is designed to throw this Error so that the problem can be debugged and taken care of.

One of the cases where this error emerges abundantly is when some code was creating numerous amounts of temporary objects and numerous weakly referenced objects in a memory-constrained environment.

Solution to this problem is found via the following ways

1. If the application has a small data set (up to approximately 100MB), then
   * select the serial collector with -XX:+UseSerialGC.
2. If the application will be run on a single processor and there are no pause time requirements, then
   * let the VM select the collector, or
   * select the serial collector with -XX:+UseSerialGC.
3. If (a) peak application performance is the first priority and (b) there are no pause time requirements or pauses of one second or longer are acceptable, then
   * let the VM select the collector, or
   * select the parallel collector with -XX:+UseParallelGC and (optionally) enable parallel compaction with -XX:+UseParallelOldGC.
4. If response time is more important than overall throughput and garbage collection pauses must be kept shorter than approximately one second, then
   * select the concurrent collector with -XX:+UseConcMarkSweepGC. If only one or two processors are available, consider using incremental mode, described below.

**Tweaking made to the PostgreSQL**

Approcah: -XX:-UseGCOverheadLimit

**Changes 1: (in posgresql.conf server side)**

**Status: unresolved)**

* change work-mem = 2M(before 1M)
* change effective-cache-size = 18G (before 128MB)
* change max-connections =10 (before 100)
* change shared\_buffers= 1G (before 128MB)

**Changes 2: (in eclipse run config )**

1. Increase heap size using Xmx switch e.g. -Xmx512m
2. Disable the error check using -XX:-UseGCOverheadLimit
3. This approach will only result in another kind of error messages – heap related java.lang.OutOfMemoryError.
4. Add -XX:+HeapDumpOnOutOfMemoryError to JAVA\_OPTS and analyse the dump with a profiler e.g. IBM Memory Analyzer [Ref](part of [IBM Support Assistant Workbench](http://www-01.ibm.com/software/support/isa/)) looking for memory leaks.
5. Change Garbage Collector Policy with -XX:+UseConcMarkSweepGC concurrent (low pause time) garbage collector (also known as CMS), or -XX:+UseParallelGC parallel (throughput) garbage collector.

Log files were generated using:

-Xloggc:gc.log -XX:+PrintGCDetails -XX:+PrintGCTimeStamps

Dump of heap files were generated using:

-XX:+HeapDumpOnOutOfMemoryError

link:

<http://www.jvmhost.com/articles/what-is-java-lang-outofmemoryerror-gc-overhead-limit-exceeded>

<http://javaeesupportpatterns.blogspot.co.uk/2011/09/gc-overhead-limit-exceeded-java-heap.html>

<http://www.revsys.com/writings/postgresql-performance.html> how to change the server parameters

[Changes done to the server?]

==================================================

* change work-mem = 16M(before 1M)
* enable maintenance-work-mem = 65MB (16MB before)
* change efficitive-cache-size = 18G (before 128MB)
* change max-connectins =10 (before 100)
* change shared\_buffers= 4G (before 128MB)
* enable vacuum-cost-delay= 50(before 0)
* enable wal-buffers

turn off auto commit

<http://www.revsys.com/writings/postgresql-performance.html> how to change the server parameters

===============================

<http://www.postgresql.org/docs/9.2/static/populate.html#POPULATE-WORK-MEM>

Increasing the maintenance\_work\_mem configuration variable when loading large amounts of data can lead to improved performance as well. This will help to speed up CREATE INDEX commands and ALTER TABLE ADD FOREIGN KEY commands. It won't do much for COPY itself, however this strategy is only useful when one or both of the above techniques were implemented.

Increasing the checkout\_segments [ref] configuration variable can also make large data to load faster. The reason is, loading a large amount of data into PostgreSQL will cause checkpoints to occur more often than the normal checkpoint frequency (specified by the checkpoint\_timeout configuration variable). Whenever a checkpoint occurs, all dirty pages must be flushed to disk. By increasing checkpoint\_segments temporarily during bulk data loads, the number of checkpoints that are required can be reduced. This eventually results in a faster load performance.

*[ref http://www.postgresql.org/docs/9.2/static/runtime-config-wal.html#GUC-CHECKPOINT-SEGMENTS]*

*useful website:*[*http://www.varlena.com/GeneralBits/Tidbits/perf.html*](http://www.varlena.com/GeneralBits/Tidbits/perf.html)

[*http://www.varlena.com/GeneralBits/Tidbits/annotated\_conf\_e.html*](http://www.varlena.com/GeneralBits/Tidbits/annotated_conf_e.html)

*shared-buffer:*

* *Start at 4MB (512) for a workstation*
* *Medium size data set and 256-512MB available RAM: 16-32MB (2048-4096)*
* *Large dataset and lots of available RAM (1-4GB): 64-256MB (8192-32768)The shared buffers parameter assumes that OS is going to cache a lot of files and hence it is generally very low compared with system RAM. Even for a dataset in excess of 20GB, a setting of 128MB may be too much, if you have only 1GB RAM and an aggressive-at-caching OS like Linux.*

**More Issues of Memory Leak**

1. Java heap space

The heap does not have a lot of space to store objects. New objects better be out of the heap to preserve heap space to run the program smoothly.

Fix: Configuration file/settings

For long running apps, app is holding references to objects and it is preventing Garbage Collector to do its job.

Another way: If code has lots of finalisers. If finals are being used, obje ct is garbage collected later. So more chances of getting outOfMemoryError

Can be monitored via: JConsole Utility

2. PermGen space

Permanent generation is full. It is the area of heap where class & method objects are stored. Basically it is the area where data of virtual machine is being kept.

Fix: Increasing -XX:MaxPermSize helps.

3. Requested array size exceeds VM limit

Application attempts to allocate an array that is larger than the heap size. If app attempts to make an array of 1Gb but max size allowed is 500Mb, OutOfMemoryError is being thrown

4.Out of swap space

- Allocation from native heap failed and native heap almost full.

- Operating system does not have enough ‘swap space’

- Another process is eating up all the memory

- Native leak: application is continuously allocating memory but it is not releasing to the Operating system after the work is done.

5.Exception

Fatal Error log has lots of information about this type of error. It happens with native code because they don’t check for returned errors after memory has been allocated.

Whats the bottlenecks of model persistence.

### About blueprints

1. It’s the bottlenecks as it takes so many memory
2. In blueprints, how the graph database to save the data:🡪DFS/BFS

(binary search/linear search), bigo notation, graph algorism.

find one db, and find out the algorism they use and maybe there is another algorism is better than that one.

The algorism the blueprint use to create graph is DFS (create all the nodes first, then add reference), there is another way to create graph called BFS (create one node and add all the references and create new node)

[repeat baseline to get updated results]

## 4.3 Provide relative solutions (Prototype)

Java performance: Reducing time and space consumption

<http://www.itu.dk/people/sestoft/papers/performance.pdf>

1. Improve performance of Java code

For loops: remove the

String: not use new word

Object:

Evaluate: this way to improve performance is good or not, are they required here or not

1. Multiple thread to read the file and crate the nodes

(How to make sure the graph is consist. For example: Thread A create node 1, 2, thread B create node 3, 4, but the node 3 connect node 4, who will create the reference? )

????????????????

Evaluate: this way to improve performance is good or not, are they required here or not

1. Loading a zip file is faster than loading a compressed file

Examples of sesame load zip file in terminal

Evaluate: this way to improve performance is good or not, are they required here or not

1. Java code to byte code: complier??????

Assume that you have a program MyApp.java and you want to run it. To execute this program you need to first compile it with javac, the JDK's built-in static Java language-to-bytecode compiler. Based on the Java code, javac generates the corresponding executable bytecode and saves it into a same-named class file: MyApp.class. After compiling the Java code into bytecode, you are ready to run your application by launching the executable class file with the java command from your command-line or startup script, with or without startup options. The class is loaded into the runtime (meaning the running Java virtual machine) and your program starts executing.

That's what happens on the surface of an everyday application execution scenario, but now let's explore what *really* happens when you call that java command. What is this thing called a *Java virtual machine*? Most developers have interacted with a JVM through the continuous process of tuning -- *aka* selecting and value-assigning startup options to make your Java program run faster, while deftly avoiding the infamous JVM "out of memory" error. But have you ever wondered why we need a JVM to run Java applications in the first place?

Evaluate: this way to improve performance is good or not, are they required here or not

1. Hadoop MapReduce ???

Summary: In this paper we will use point 2 and point \* to implement our own prototype and get our new result.

## 4.4 New Result:

**OrientDB result:**

|  |  |  |  |
| --- | --- | --- | --- |
| Set0 |  |  |  |
| Native |  |  |  |
| blueprints |  |  |  |

C.C. Aggarwal and H. Wang. Managing and mining graph data, volume 40. Springer, 2010.

5. Results

Analysis the result

1. Conclusion

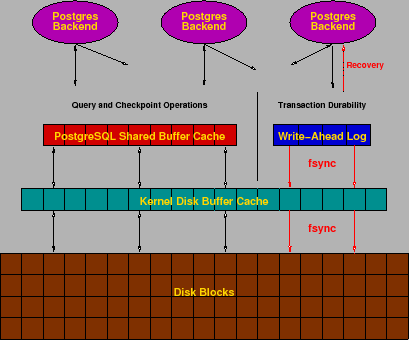
What the basic work

Future work for phd

ps aux

# //how to change the server side: why not working

# POSTGRESQL Shared Buffer Cache

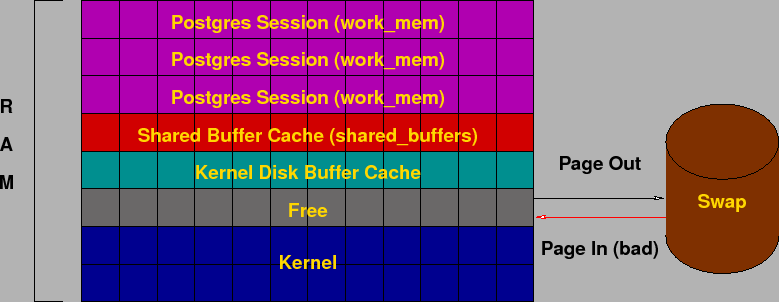


POSTGRESQL does not directly change information on disk. Instead, it requests data be read into the POSTGRESQL shared buffer cache. POSTGRESQL backends then read/write these blocks, and finally flush them back to disk.

Backends that need to access tables first look for needed blocks in this cache. If they are already there, they can continue processing right away. If not, an operating system request is made to load the blocks. The blocks are loaded either from the kernel disk buffer cache, or from disk. These can be expensive operations.

The default POSTGRESQL configuration allocates 1000 shared buffers. Each buffer is 8 kilobytes. Increasing the number of buffers makes it more likely backends will find the information they need in the cache, thus avoiding an expensive operating system request. The change can be made with a *postmaster* command-line flag or by changing the value of *shared\_buffers* in *postgresql.conf.*

# How Big Is Too Big?



You may think, ``I will just give all my RAM to the POSTGRESQL shared buffer cache.'' However, if you do that, there will be no room for the kernel or for any programs to run. The proper size for the POSTGRESQL shared buffer cache is the largest useful size that does not adversely affect other activity.

To understand adverse activity, you need to understand how UNIX operating systems manage memory. If there is enough memory to hold all programs and data, little memory management is required. However, if everything doesn't fit in RAM, the kernel starts forcing memory pages to a disk area called *swap.* It moves pages that have not been used recently. This operation is called a *swap pageout.* Pageouts are not a problem because they happen during periods of inactivity. What is bad is when these pages have to be brought back in from swap, meaning an old page that was moved out to swap has to be moved back into RAM. This is called a *swap pagein.* This is bad because while the page is moved from swap, the program is suspended until the pagein completes.

Pagein activity is shown by system analysis tools like *vmstat* and *sar* and indicates there is not enough memory available to function efficiently. Do not confuse swap pageins with ordinary pageins, which may include pages read from the filesystem as part of normal system operation. If you can't find swap pageins, many pageouts is a good indicator you are are also doing swap pageins.

**Conclusion**

Basic things u found out

Summarize whole document

[blueprint was a solution, but it has lots of bottlenecks]

indentify baseline

Future works

Wht to do in phd