U.P.B. Sci. Bull., Series C, Vol. 72, Iss. 4, 2010 ISSN 1454-234x

EXPERIMENTAL RESULTS ON THE PERFORMANCE OF A NEW CONTENT MANAGEMENT MODEL

Rareş VASILESCU[1]

Sistemele de gestiune a conţinutului folosesc metode diverse de stocare şi gestionare a informaţiei. Volumul informaţiei creşte într-un ritm exponenţial iar sistemele de gestiune a conţinutului încep să fie des folosite ca platforme pentru dezvoltarea aplicaţiilor. Aceste condiţii generează nevoia existenţei unor soluţii din ce în ce mai performante. În această lucrare se va prezenta un set de teste experimentale realizate asupra unui nou model de sistem, model proiectat astfel încât să ofere performanţă maximă atât din punct de vedere funcţional cât şi non-funcţional.

Content management systems use various strategies to store and manage information. Information volume increases at exponential rate and content management systems become more and more a platform used by applications to provide services to users. Given these conditions there is a growing need for high performance content management systems. In this paper we present some tests done on a new content management system model designed to provide maximum performance both in terms of functional and non-functional requirements.

Keywords: content management, performance, tests, experiments

1. Introduction

Content management systems (CMS) can be defined as a set of processes and technologies which support the digital information management lifecycle. This digital information is usually referred as “content” and can be found as not-structured or semi-structured - such as photographs, images, documents or XML data.

The size of the managed content generates specific challenges for CMS, as it can range from several bytes to hundreds of gigabytes. Building an informational system to manage such data is a real challenge, especially if we take into consideration that these elements need to be processed securely in highly concurrent multi-user environments.

A new model of a high performance content management system was proposed [1] and this paper presents experimental results which show the model performance and the possible areas for improvement.

[1] Eng., Computer Science and Engineering Department, University POLITEHNICA of Bucharest, Romania, e-mail: raresv@yahoo.com

The proposed model shows a content management system which stores data in an autonomous, self descriptive manner, scalable both in terms of functionality and of usage. Individual content items are self-described and stored in a standardized format on generic file systems.

Comparing with the traditional models, this new architecture does not rely on a relational database management system but defines a specialized indexing and retrieval mechanism. This mechanism addresses the specific needs of content management and focuses on delivering fast responses to user queries while also providing a high degree of flexibility and long term data persistence capabilities.

In summary, indexing agents are implemented to store and manage each attribute defining the content and search agents are put in place to respond to use queries. All agents work in parallel and are governed by a monitor which distributes the processing tasks and then collects and aggregates the results as needed.

Aiming for high performance, it is important to know what is the behavior of this model therefore an implementation was done and some initial experiments conducted. This paper presents the results of these experiments.

2. Performance evaluation concept

One important topic on performance evaluation is to measure the system processing capacity while applying a certain load and monitoring the used resources.

In performance evaluation one must also take into consideration the other, non-quantitative system characteristics, since the overall performance of a model is highly influenced by its qualitative attributes.

We need to establish an independent set of metrics and procedures so that we could obtain a relevant indication of the observed system performance. Such procedures exist for relational database management systems (such as AS3AP, Wisconsin or TPC benchmarks [2]) but these do not properly address the specialized area of data management focused on content management. Evaluation methods included in these benchmarks are focused on transaction and data processing without taking into account the specific data access patterns of content management, thus we should not apply these tests on the new model.

The paper will present the experimental results obtained by applying a set of classic tests on the implementation:

- Data ingestion
- Data update
- Data retrieval

In order to obtain an indication whether or not the proposed data indexing and retrieval strategy provides a performance increase over the traditional model, two indexing and retrieval strategies were implemented : one using an index structure optimized for in-memory processing and one using a traditional relational database management system. The independence to other factors was assured by the architectural design of the model itself, which enables usage of various indexing and retrieval strategies on the same data.

The hardware environment was represented by a classic workstation with a single Intel 32 bit processor and under 2 GB or RAM available on Windows XP and Windows Vista operating systems. The input/output hardware and software subsystem was a reduced performance one, with no specific optimization.

Since the model implementation was done completely from scratch, we had the opportunity of implementing measurements directly inside the source code. This allowed precise measurement of key sub-parts of the model not only a black box observation.

Environmental measurements were done also on the system resources during tests so that the overall impact on CPU, disk and memory can be observed. The results presented in this paper come from single-user tests but using the highly parallel processing techniques and without implementing think-times usually found in such benchmarks. This decision was made so that a baseline can be established for further experiments.

3. Performance evaluation for data load

Content management systems are frequently used for archiving large volume of information. For example, such systems are used for long term archiving of all email messages within an organization. As a case study, if an organization has around 1.000 employees and each of them sends or receives an average of 50 messages daily this sums up to 50.000 new complex information objects each day – over 12.5 million objects each year. If we consider a retention period of 10 years this means the content management system must be able to manage at least 125 million objects. More, the 50.000 daily objects would be produced in approximatively 8 hours which means an average of 2 objects per second. Given this average we can assume that there are peaks when the system must handle tens of new complex objects every second.

We need to measure the performance of the CMS in this context; therefore we aim to determine whether or not the new model can ingest such a large volume while it’s also managing the historical data.

To simulate the highest possible load the test was conducted with a maximum throughput without including thinking time delay (which is usually included in data batches to simulate human operators).

Load tests were conducted for various quantities: 1.000, 10.000, 100.000 and 1.000.000 objects. Each test was repeated several times, alternating with other tests to obtain relevant average values. These iterations also generated a large volume of objects inside the content management system which provided a very good test bed for analyzing the scaling capabilities.

Each newly inserted object had 5 specific metadata attributes of different types (number, date and time, character strings) and content in size of 1 kilobyte, resulting in about 1200 bytes for each stored object.

During each test the following metrics were monitored:

- Average time to permanently store an object
- Average time to index a metadata attribute
- The instantaneous size of the indexing queue

First test was the insertion of 1.000 objects. The test took an average of 7 seconds.

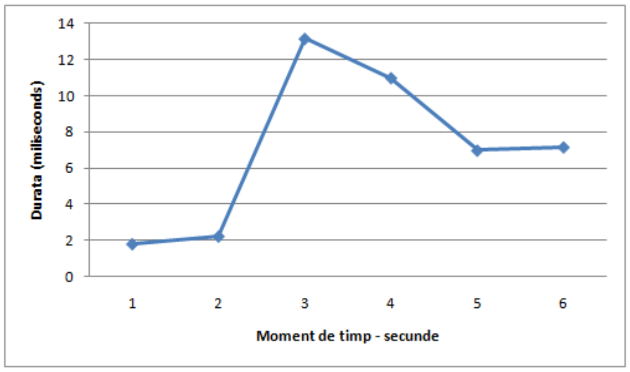

Fig. 1. Object storage time – test for 1.000 new objects

The average time to store an object is shown in Fig 1. As observed the time oscillated between 2 and 13 milliseconds. At the beginning the insertions were typically faster but as average the time needed to store the test objects was about 7 milliseconds.

While the objects are stored they need also to be indexed (their associated metadata) so that they are available to search queries. We measured the time needed to process the index information and the results are presented in Fig 2.

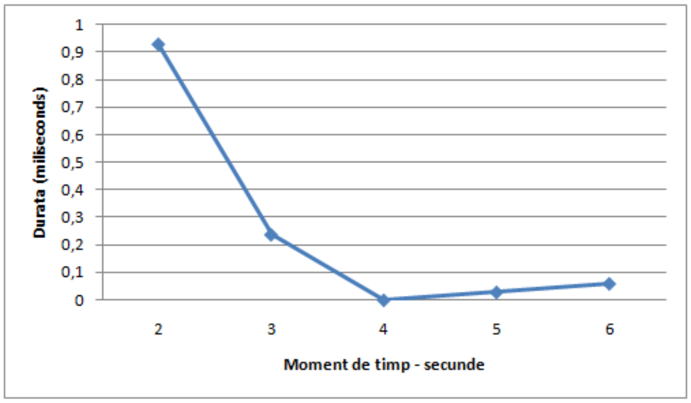

Fig. 2. Object information index time – test for 1.000 new objects

The time needed to index the information (5.000 index values corresponding to the 1.000 new objects) was about 5 seconds and we can observe that after the initial insert (measured as at a little under 1 millisecond in average) all subsequent operations were significantly faster (one magnitude order lower). This is explained by the time needed to initialize and open the index structure while after its related data pages were loaded up in main memory the subsequent operations were significantly faster since they were no longer I/O bound.

Care was taken to analyze whether or not the "commit" operation has any impact on the overall performance. The tests were performed with the index structure persisting information after each insert and persisting whole information only at the end of the load respectively. No significant time difference was identified (the time needed to index an attribute varied within the same limits in both cases).

The index queue was monitored to see if the index subsystem is slower or faster than the persistent object storage subsystem. For the given test data the index queue was almost all the time near 0 which means that for a batch of 1.000 new objects the indexing system is at least equivalent if not faster than the storage subsystem. This is normal and expected behavior at this stage but further tests are needed to demonstrate the same ability when the system is already loaded with a large volume of information.

The next chosen step was to perform the same tests with a volume ten times bigger (10.000 new objects).

Storing 10.000 new objects lasted approximatively 3 minutes. The average time needed to store an object was measured at 15 milliseconds. This value is consistent with the results obtained for the 1.000 objects test, although is positioned at the higher end of the initial measures.

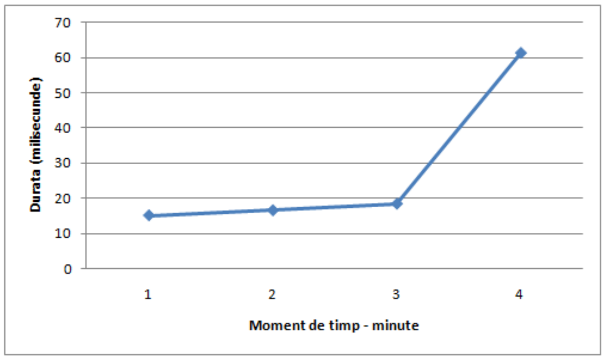

Fig. 3. Object storage time – test for 10.000 new objects

The test showed a slight increase of the average time during its execution and spiked at the end. Since this dime is directly influenced by the performance of the I/O infrastructure we consider that the large volume of objects reached a limit of the I/O operating system cache and therefore a lot of cache misses were generated at the end.

As in the previous runs, the time needed to index an information element was also measured.

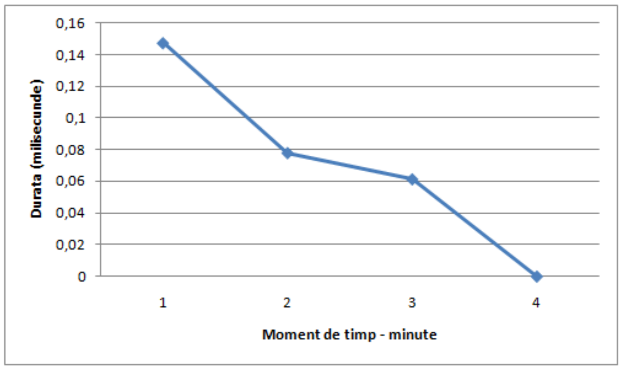

Fig. 4. Object information index time – test for 10.000 new objects

Measured indexing time varied quite a lot but was actually aligned with the performance recorded in the first runs (under 0.1 milliseconds per attribute).

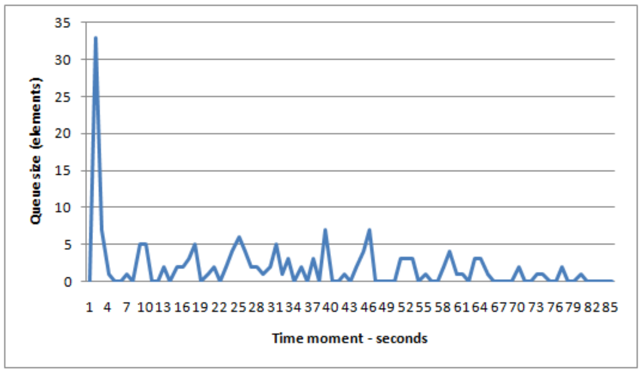

Fig. 5. Index queue size – test for 10.000 new objects

The index queue size was big in the beginning, corresponding with the large index time associated with the test run. As the indexes became loaded in memory the queue was keeping up with the load (rarely exceeding 5 items which is exactly the number of queue items generated by one complex object).

Performing test runs with 100.000 new objects was the text step in the performance evaluation. Such runs lasted around 24 minutes each. Consistently with the previous runs (e.g. the 10.000 object ones), object storage time kept around 15 milliseconds per object.

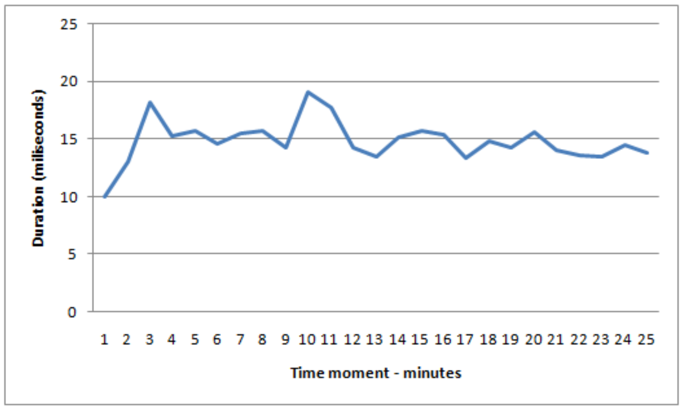

Fig. 6. Object storage time – test for 100.000 new objects

The index performance was also very similar with the previous runs; an average value of 0.08 milliseconds was recorded over the 500.000 processed attributes. The measured deviation around this average was of maximum 0.06 milliseconds.

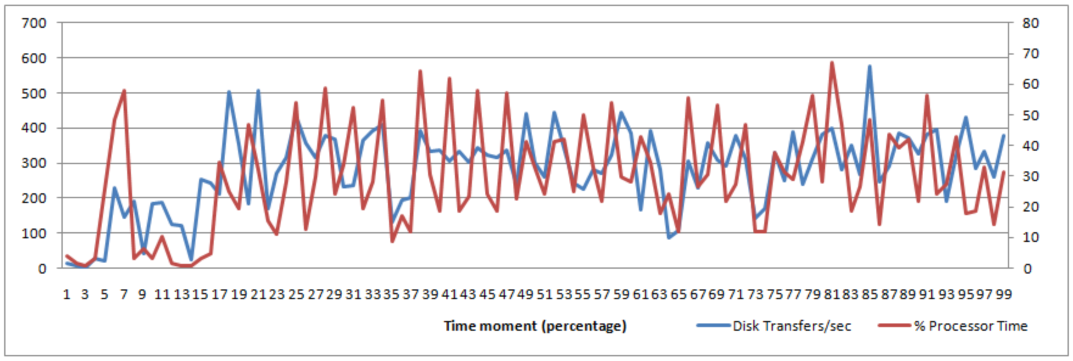

Fig. 7. System load (CPU and I/O) – test for 100.000 new objects

Looking at the measurements of the system resources during the test period we can observe that there was no CPU overload (the CPU time averaged at 35% with occasional spiked no larger than 60%). We also see that there is a correlation between the disk operations and processor activity – both seem to be following the same pattern, thus we can determine that the architecture is not processor intensive but was I/O bound during the tests.

The indexing queue size evolution shows that the index processing kept up perfectly with the load – the size of the queue was constantly under 5 values.

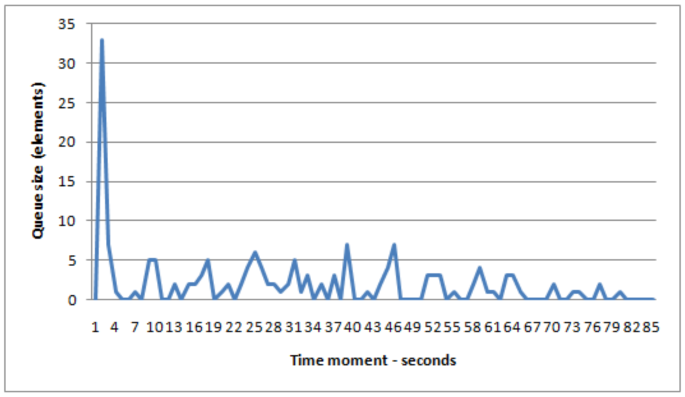

Fig. 8. Indexing queue size during test – run for 100.000 new objects

One can notice that even though the queue started with a massive size at the beginning of the test, it recovered very quickly and within the first minute it reached a normal (almost empty load). The initial load of the queue was generated by the fact that the indexing engine had not fully initialized while the new items were pushed into the system – with no agents able to pick up the queue items its size spiked upwards. Once the agents were initialized they quickly processed the queue and were able to keep up with the continuous load.

Based on this evidence we can conclude that the indexing engine seems to have a very high capacity of absorbing information.

The endurance test of the system was to repeatedly add millions of objects to it – which lead to a repository summing around 10 million objects.

On average the tests shown that 1.000.000 objects were ingested in 4 hours and 50 minutes, leading to an average of 150 objects per second.

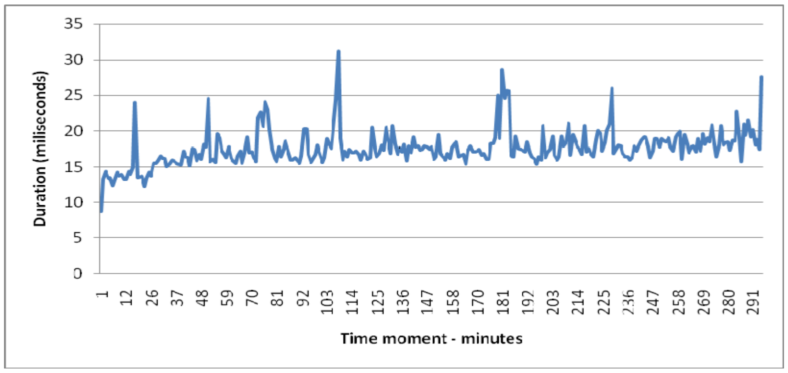

Fig. 9. Object storage time – run for 1.000.000 new objects

The average time needed to persistently store one object averaged between 15 and 20 milliseconds, with spikes at some moments in time – probably due to external operating system filesystem actions.

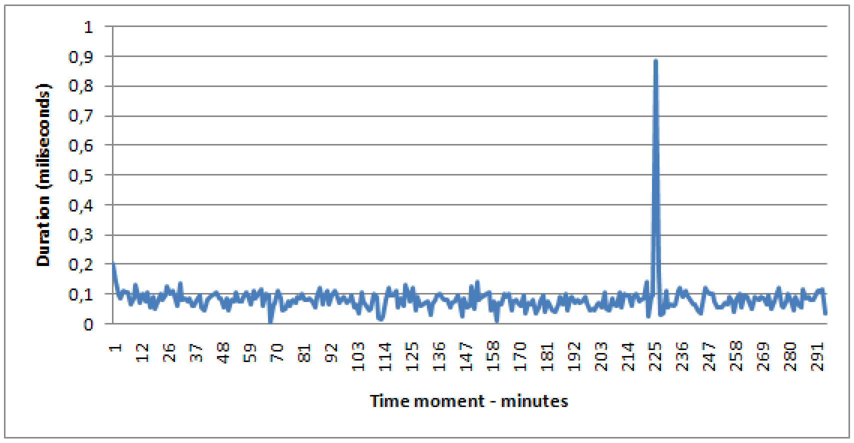

Fig. 10. Object information index time – run for 1.000.000 new objects

The average time needed to index an information element was measured at around 0.1 milliseconds. On one run we noticed a sudden and brief drop of performance (for less than 1 second) in which the average indexing time neared

0.9 milliseconds (measured at 0.88 ms). This spike could not be explained by other correlated measures and it's cause yet to be determined and requires more experiments. These measurements are consistent with the ones found for the 100.000 objects runs; therefore we can conclude that the system performance seems to not visibly degrade during extensive load.

The same consistency was observed when measuring the indexing queue size. This measure averaged at fewer than 2 elements. As in other smaller tests, there was in initial warm-up time needed by the indexing agents to load and begin handling the load. In this case the queue topped at about 1.000 items and then was completely processed in one second as the agents started processing.

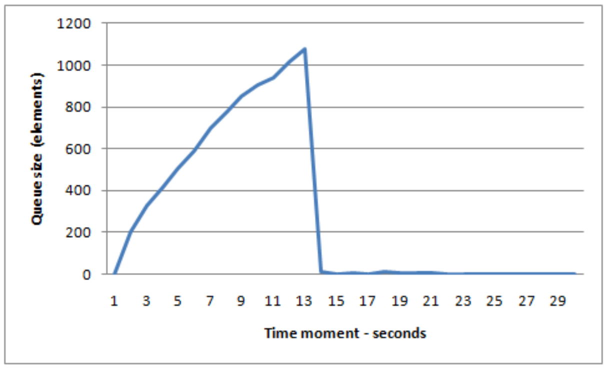

Fig. 11. Indexing queue size during test – run for 1.000.000 new objects – first 30 seconds

This behavior is encouraging and supports the taken architectural decisions. It seems that the decisions lead to a robust system which is able to manage large information volumes.

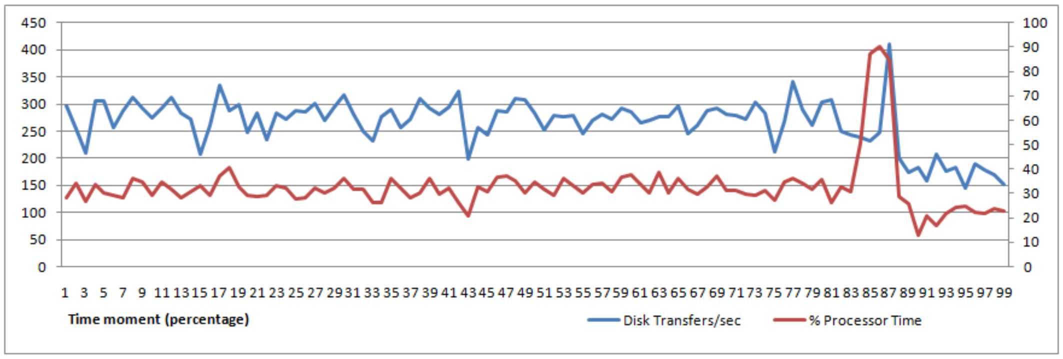

Fig. 12. System load (CPU and I/O) – test for 1.000.000 new objects

Observing the system parameters (CPU load and input-output activity) we see that the correlation between these two indicators remained the same as in previous test runs. CPU load averages at 32% and I/O operations per second averages at 265.

Alternative indexing strategies

The indexing engine used in the previously monitored tests implemented the strategy to use the available memory to the maximum and limit the access to I/O subsystem. Index agents were implemented using in-memory indexes adapted also for permanent storage so that they are not limited to only the available memory.

A specific feature of this architecture is that it allows implementation of various index implementation techniques. We will leverage this advantage by plugging in the system a new index strategy which uses a standard relational database management system. We chose a commercial RDBMS system with low performance in order to be able to easily point any differences.

We performed all the tests described above, starting with the load of 10.000 new objects.

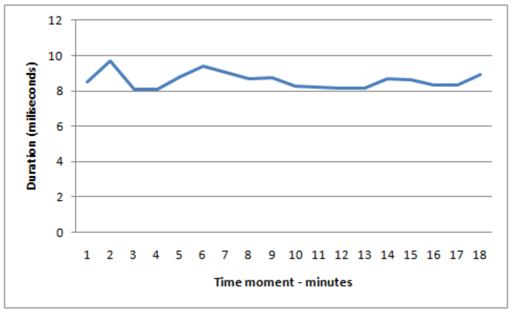

Fig. 13. Object storage time – test for 100.000 new objects (rdbms)

The time needed to store an individual object was a constant for the duration of the tests. Some tests experienced a drop in performance at the end which did not impact the overall results but still looks extremely familiar with the event which happened for the other indexing approach and thus it will need more tests and analysis.

The measured average time to store an object was around 10 milliseconds which is actually very similar with the measures for other implementation. This makes sense since the object storage time does not depend on the indexing strategy, which is now experimentally confirmed.

Next step is to look and see what the average indexing time of one information element is.

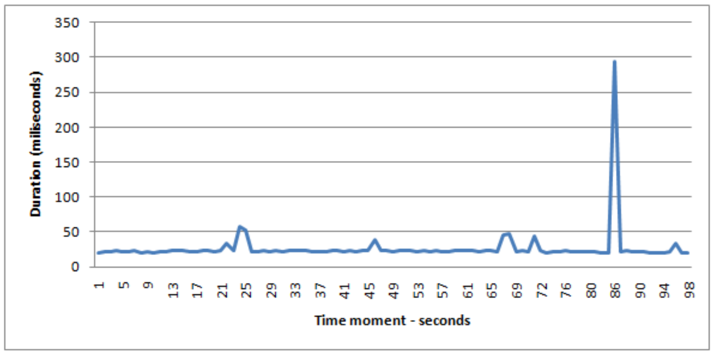

Fig. 14. Object information index time – run for 10.000 new objects (rdbms)

The average time to index an information element was about 22 milliseconds. This time is almost 200 times bigger than the one experienced for the primary indexing technique.

We still observe also the performance drop spike which is correlated with the object storage spike. Since both the object storage and the indexing processing are I/O bound a possible conclusion would be that at that precise moment an externally triggered I/O operation interfered with the tests.

The next measurement to look at is the evolution of the indexing queue size during the test run.

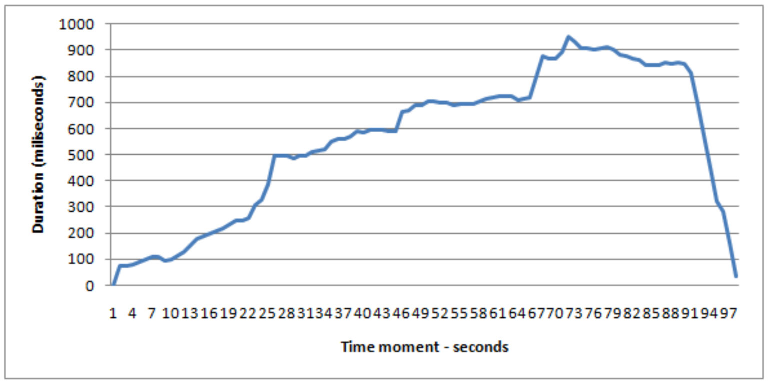

Fig. 15. Indexing queue size during test – run for 10.000 new objects (rdbms)

This is probably the most relevant measurement for this experiment. It clearly shows that the indexing engine could not keep up with the load given to the system – the queue keeps growing as the load keeps coming.

The queue size drops only in the last 7 seconds, time in which no new objects were created and thus the indexing engine could recover. The main conclusion is that the RDBMS based system could not handle the load the other strategy was very good at processing.

Going back to the much bigger test (with 100.000 objects) we can observe the same behavior of the indexing queue. In this case the queue needed about 3 minutes to process the backlog (vs. 7 seconds for the 10.000 objects run).

We also observed that the average time needed to index one piece of information remained at the previously measured average of 22 milliseconds.

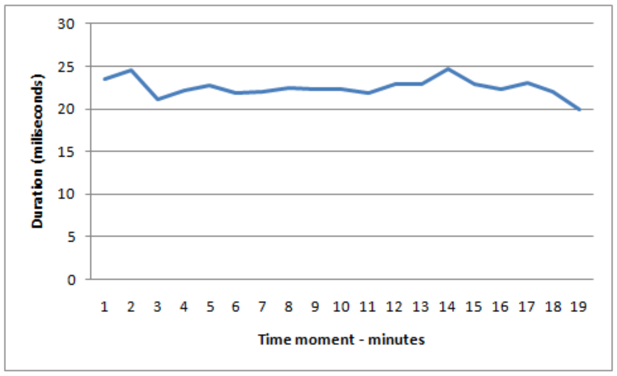

Fig. 16. Object information index time – run for 100.000 new objects (rdbms)

One main conclusion we take after these experiments is that choosing the indexing strategy makes a very big different in the performance of the newly designed content management architecture. As of now the best identified strategy was to keep the index structure in memory as much as possible using a LRU (Least Recently Used) model and structuring the information by "columns" instead of "rows".

4. Conclusions

These experiments showed that the design can lead to extremely high performance but care must be taken in choosing the adequate indexing strategy. While tests seemed to reveal that the memory focused indexing structures combined with a vertical approach on managing metadata display a very good performance, we need to take caution and not dismiss other indexing techniques. More tests and experiments are needed to validate these initial results.

As next steps we aim to enhance the current implementation with an international standard for interoperability between content management systems – CMIS [3]. When this implementation is ready the system will be able to be tested in comparison with other similar systems thus marking a significant step in evaluating its performance.

REFERENCES

[1] *R. Vasilescu*, An Alternative to Common Content Management Techniques, IJCSIS 2009, **Vol. 6**, No. 1, pp. 056-060, October 2009, USA

[2] *M. Petrescu, R. Vasilescu, D. Popeanga*, Performance Evaluation in Databases – Analysis and experiments, Fourth International Conference on Technical Informatics CONTI'2000, 12-13 October, "Politehnica" University of Timisoara

[3] OASIS, "Content Management Interoperability Services (CMIS) TC", 01.04.2009, http://www.oasis-open.org/committees/cmis, accessed on 01.11.2009.