Proceedings of 2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, 13-16 May 2013, Delft, Netherlands, which should be cited to refer to this work.
DOI: 10.1109/CCGrid.2013.33

Predictive caching in computer girds

Efstratios Rappos
Institute for Information and Communication Technologies
HEIG-VD
Yverdon-les-Bains, Switzerland
Email: efstratios.rappos@heig-vd.ch

Stephan Robert
Institute for Information and Communication Technologies
HEIG-VD
Yverdon-les-Bains, Switzerland
Email: stephan.robert@heig-vd.ch

***Abstract*—We present a model for predictive caching where a shared cache is used to improve performance across a grid. Unlike local caching mechanisms, shared, grid or cloud-based caches incur high costs or latency associated with the additional data transfer. Our proposed caching model, which is dynamically optimized and constantly updated over time, determines the optimal allocation of objects into the shared cache, in such a way that the total cost or latency is minimized. This is achieved by including in the caching algorithm design measures of grid latency, data retrieval costs and a predictive component based on the probability of cached objects being requested in the near future.**

***Keywords*-cache storage; grid computing; mathematical programming**

I. Introduction

Caching constitutes a fundamental mechanism for improving performance in a variety of networked computer systems and applications. The importance of caching in a cloud or grid environment has been highlighted by several researchers [1], [2] and novel caching products such as Microsoft's Windows Azure Shared Caching service [3].

Recent research has focused on adaptive caching systems where the caching algorithm changes dynamically with time to improve performance [4], [5]. These methods aim to keep both the cache contents and the caching algorithm itself up-to-date over time.

We propose a self-adjusting caching method where the caching algorithm dynamically adapts itself based on a predictive model of the likelihood that objects in the cache will be requested in the future. The algorithm is particularly geared for grid or cloud environments, as it includes in its design the additional costs associated with distributed cache systems.

II. A predictive caching model for grid-based systems

In the proposed model we assume that a grid of computers accesses a database which uses a shared cache. One of the key challenges in designing the cache management algorithm is the increased cost of data transfers associated with shared grid-based cache systems. In our model we include these costs in the caching algorithm design, so these costs affect the decision on which elements to include in the cache and which to eject.

The mathematical model behind the cache updating algorithm works in discrete time, and at each time period the following optimization problem is updated and solved. The solution of this mathematical model, presented below, determines which elements to include in the cache for the next period.

$$\text{minimize} \sum_i c_{1i}p_ix_i + c_{2i}p_i(1-x_i) + c_{3i}x_i \qquad (1)$$

$$\text{subject to } \sum_i s_ix_i \leq C \qquad (2)$$

$$x_i \in \{0,1\}. \qquad (3)$$

The binary variable x_i determines whether the object i will be placed in the cache ($x_i = 1$) or not ($x_i = 0$). The probabilities p_i in the objective function (1) represent an estimate of the probability that the object i will be requested in the next time period. These probabilities are updated at each time period which is achieved by looking past object accesses and calculating the likelihood that each item will be requested next. The parameters c_1, c_2 and c_3 present the various cost elements. In particular, c_{1i} is the estimated cost or latency of obtaining object i from the shared cache, c_{2t} is the cost object i directly from the data server and c_{3i} is the cost to place object i in the shared cache. The model constraint (2) is that the total size s_i of all elements chosen to be placed in the cache does not exceed C, the maximum shared cache capacity.

The above model is similar to knapsack problems in optimization [6] and can be easily solved using both exact combinatorial optimization techniques or fast approximation heuristics.

III. Simulation and numerical experimentation

The proposed model was tested in a simulation environment. We compared the performance of a virtual shared cache using the predictive caching model against an identical setup which uses the least-recently-used (LRU) algorithm or the least-frequently-used (LFU) algorithm. For comparison reasons we also report on the performance of the Belady optimal caching algorithm [8], which is the theoretical (a posteriori) optimal caching method among algorithms that cache the last requested object. In the course of the simulation, we recorded the number of cache hits and the cost per cache hit for each of these algorithms for cache maintenance.

The data request sequence, consisting of 10,000 requests, was randomly generated from a Zipf distribution with a Zipf exponent $\gamma = 1$, 1.5, 2, 3 or 4. The remaining parameters of the model were chosen as follows: the set of distinct data objects that can be requested consisted of 100 objects, and the size of each object was randomly chosen in the range 50–150. The cache capacity used was 1,000, so on average around 10 objects will fit into the cache at each time.

The costs were based on the object size (with an added random 'noise' in the range $\pm 5\%$) but the following relationship between the costs was used, based on assumptions used in [7]: $c_1 = 50c_2$ and $c_2 = 100c_3$. The probabilities p_i were initialized to $1/100$ and adjusted at each step according to the frequency an object was requested in the previous steps.

The results of this preliminary computational experimentation are shown in Table I. We note that the predictive caching model results in up to 46% more cache hits compared to LRU and up to 26% more cache hits compared to the LFU algorithm. The cost per cache hit was also improved by up to 43% and 30% compared to the LRU and LFU algorithms respectively.

We observe that in some cases the predictive algorithm outperforms the Belady algorithm; this is because Belady is the theoretical optimal algorithm among those who cache the last requested object. However, in a shared cache environment it is not always true that caching the last object is optimal, if for example the object has a low probability of being requested again and instead prefetching a different object may be more beneficial.

Table I
COMPARISON OF SHARED CACHING METHODS

Zipf workload with parameter γ	*Predictive*	*LRU*	*LFU*	*Belady*
Cache hits				
$\gamma = 1$	5748	3927	4551	6112
$\gamma = 1.5$	8320	7418	7466	8427
$\gamma = 2$	9516	9076	9008	9458
$\gamma = 3$	9957	9912	9805	9945
$\gamma = 4$	9990	9977	9952	9982
Cost per cache hit				
$\gamma = 1$	154.3	271.5	221.3	138.7
$\gamma = 1.5$	66.9	87.9	86.8	65.1
$\gamma = 2$	62.9	70.3	71.6	63.6
$\gamma = 3$	39.8	40.4	42.0	40.0
$\gamma = 4$	59.2	59.4	59.9	59.4

IV. CONCLUSION

This paper presented a model for designing a shared cache updating method, where the caching strategy is determined by cost considerations of moving data across the network and a predictive element of estimates on the probability that a particular data request will be asked by grid users in the near future. The caching strategy is modeled by an optimization problem, the solution of which determines at each time period the optimal allocation of objects into the shared cache, in such a way so that the cost of caching and expected cost of future data retrievals is minimized.

The proposed caching architecture will prove useful in platforms such as remote datacenters where there is a significant overhead associated with the operation of a shared cache. In such cases, the performance of the cache can be improved by adapting the traditional algorithms used for the management of the cache in a way that takes into consideration the associated overheads and dynamically adjust the cache mechanism to ensure the most useful objects are cached.

In our experimental setup we observed an up to 46% improvement in cache hits compared to the traditional LRU and LFU algorithms and a similar improvement in the cost per cache hit.

The next steps in this research on dynamic, self-adjusting caching algorithms will include the implementation and testing of the proposed framework in a real-user cloud or grid environment, where the real-life performance and potential of the caching framework will be more accurately assessed.

REFERENCES

[1] A. W. Gordon and P. Lu, "Low-latency caching for cloud-based web applications," in *NetDB 2011*. Athens: ACM, June 2011.

[2] D. Dash, V. Kantere, and A. Ailamaki, "An economic model for self-tuned cloud caching," in *IEEE International Conference on Data Engineering*, 2009.

[3] Microsoft. (2012) Windows Azure shared caching. [Online]. Available: http://msdn.microsoft.com/en-us/library/windowsazure/hh914133.aspx

[4] S. Sarwar, Z. Ul-Qayyum, and O. A. Malik, "A hybrid intelligent system to improve predictive accuracy for cache prefetching," *Expert Systems with Applications*, vol. 39, pp. 1626–1636, 2012.

[5] W. Wang, P. Mishra, and A. Gordon-Ross, "Dynamic cache reconfiguration for soft real-time systems," *ACM Transactions on Embedded Computing Systems*, vol. 11, no. 2, p. Article 28, July 2012.

[6] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack problems*. Springer-Verlag, 2004.

[7] G. Yadgar, M. Factor, K. Li, and A. Schuster, "Management of multilevel, multiclient cache hierarchies with application hints," *ACM Transactions on Computer Systems*, vol. 29, no. 2, p. Article 5, May 2011.

[8] L. A. Belady, "A study of replacement algorithms for a virtual-storage computer," *IBM Systems Journal*, vol. 5, no. 2, pp. 78–101, 1966.