Quasar

https://www.csl.cornell.edu/~delimitrou/papers/2014.asplos.quasar.pdf

Resource-Efficient & QOS-Aware Cluster Management

By: Hanna Hayik

Introduction

Cloud Computing is one of the most popular areas in the Computer Science field in the 21st century, Public cloud vendors include: Windows Azure, Amazon’s EC2 and Google Compute Engine.
Cloud platforms offer flexibility & cost efficiency as advantages over other methods in computing.
Companies build data centers and manage them while Customers can launch computing jobs on these centers and that can range from batch jobs to interactive services.

Some of these clouds use Cluster Management, which means that, a cluster of nodes (computers/servers…) can share a specific job to better consume resources and improve execution times.
One of the biggest problems in these centers even for those using Cluster management is *Low Utilization [1, 2],* Meaning that the launched jobs are not fully consuming their resources, which lowers cost effectiveness for the cloud company.

The algorithm we are presenting, *Quasar*, Was made to increase utilization across the resources in the cloud such as CPU, Memory, Bandwidth and more. *Quasar* is a cluster manager that maximizes resource utilization while keeping performance & QoS requirements.

Algorithm

Quasar is performance based opposed to most cluster managers that are reservation based, In normal cluster managers, jobs reserve an amount of the resources they need but researches have shown that users/jobs require an exaggerated amount of resources that they will mostly not use causing low utilization. In *Quasar* users that want to launch a job can choose a performance constraint without knowing what resources are being chosen, These constraints are expressed as Queries-per-second (QPS) or latency (Quality-of-service) or execution time (for distributed frameworks like Hadoop).(Look Full table down).

Job Type	Latency-critical	Distributed framework	Single-thread	Multi-threaded	Single-node
Constraint	QPS & QoS	Execution time	IPS	IPS	IPS (instructions per second)

Quasar runs in three stages:
First after a performance constraint is entered, profiling the workload is done on a few servers, this profiling info is mixed with the information from the offline profiling done by the manger and the info from the previously scheduled jobs, together they produce accurate estimations of performance as we change the resources such as number of cores in node or interference of other workloads.

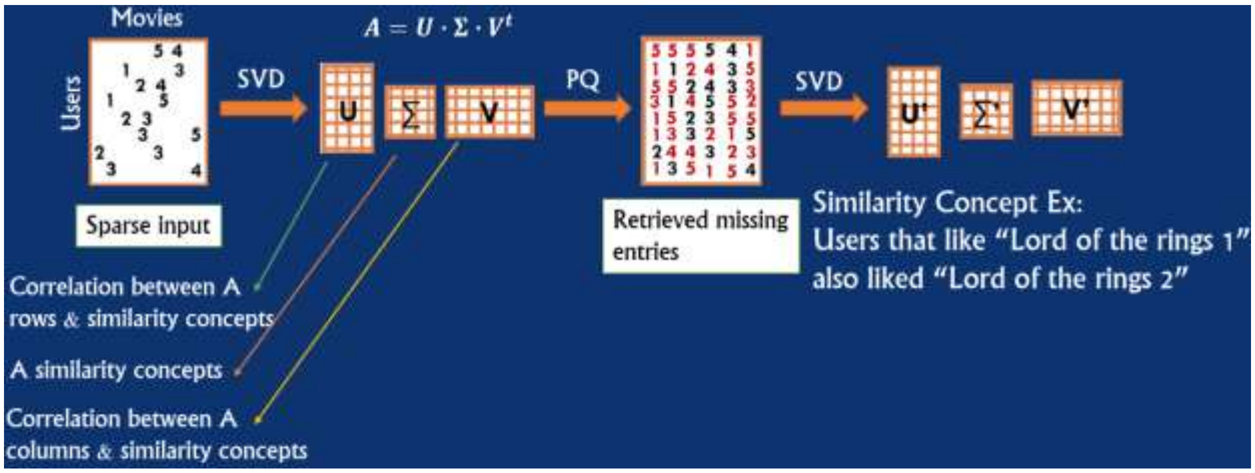
Second Collaborative filtering is used to accurately estimate the effect of four perspectives on the workload & constraints defined by the user. The four aspects of this classification stage are:

- Heterogeneity: measures the effect of server type on performance
- Interference: measures the sensitivity of sharing resources in the same node (with other workloads) on performance
- Scale-Up: measures the impact of resources used in the server (cores, RAM, storage…) on performance
- Scale-Out: measures the impact of varying number of servers on the workload performance

With these four classifications, we obtain accurate results about how/which resources we should use, but does *Quasar* try every combination of resources to obtain the optimal one? Enter
Collaborative Filtering:
is a technique used in recommendation systems with low-density inputs. After profiling on a few servers, the results are sparse matrices, which *Quasar* will use Singular Value Decomposition and PQ-reconstruction on, to fill them and obtain the full results across the four tested aspects. For example suppose we want to know the effect of Scale-Out (number of servers effect on performance): our manager will build a low-density input matrix that contains the few profiling results obtained from the first stage where the rows are apps and columns are nodes, by using Singular Value decomposition we decompose our matrix A into 3 matrices:

- U: this matrix express the correlation between A's rows and the similarity concepts
- ∑: this matrix is the similarity concepts
- V: this matrix expresses the correlation between A's columns and the similarity concepts

With the given formula: $\boldsymbol{A = U \cdot \Sigma \cdot V^t}$**,** next we perform a PQ construction from the three matrices above into the original matrix A but now we have retrieved the missing entries and we have a high-density matrix which has accurate estimated results for a specific classification. (A picture is inserted from my presentation in class to visually explain the stage).


Movies
Users
Sparse input
SVD
$A = U \cdot \Sigma \cdot V^t$
U
Σ
V
PQ
Retrieved missing entries
SVD
U'
Σ'
V'
Similarity Concept Ex:
Users that like "Lord of the rings 1" also liked "Lord of the rings 2"
Correlation between A rows & similarity concepts
A similarity concepts
Correlation between A columns & similarity concepts

Third stage is the resource allocation and assignment through a greedy algorithm, the greedy algorithm uses the 2nd stage output through ranking the available servers by decreasing quality and it expands the allocation based on available resources until the performance requirement is achieved. Despite the greediness of the algorithm, graphs show that the allocations & assignments are of high quality leading to high performance and high resources utilization.

The scheduler decides on resources according to Interference & Heterogeneity first, after that it scales-up (increases resources in the selected nodes) until performance constraint is met, last and is optional it scales-out the workload distributing it across many servers.

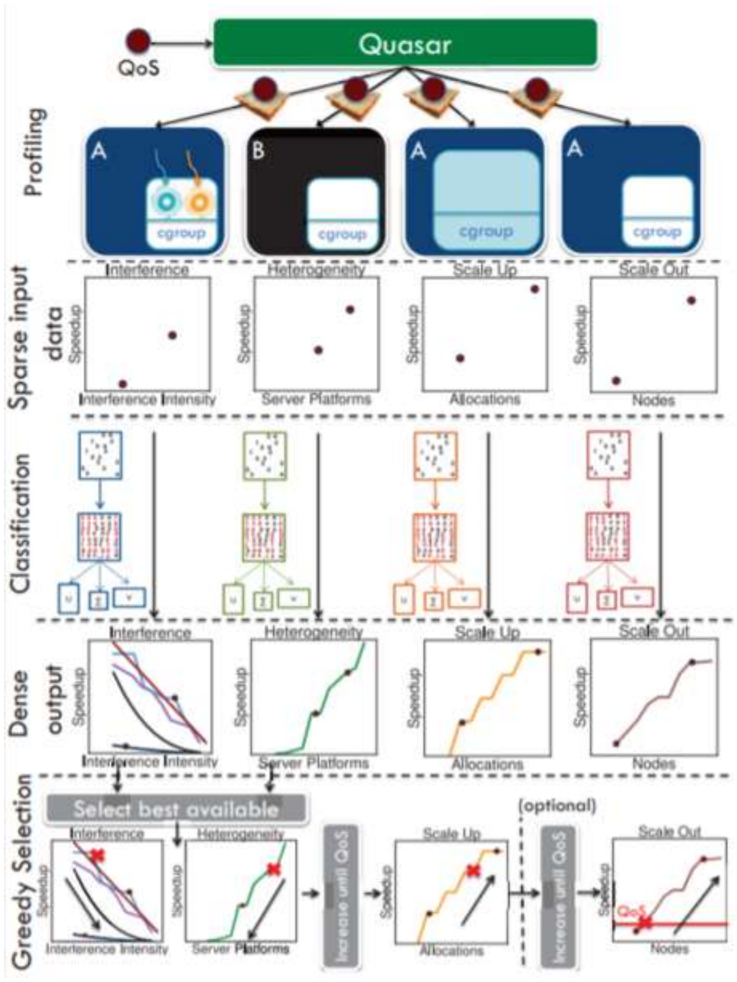
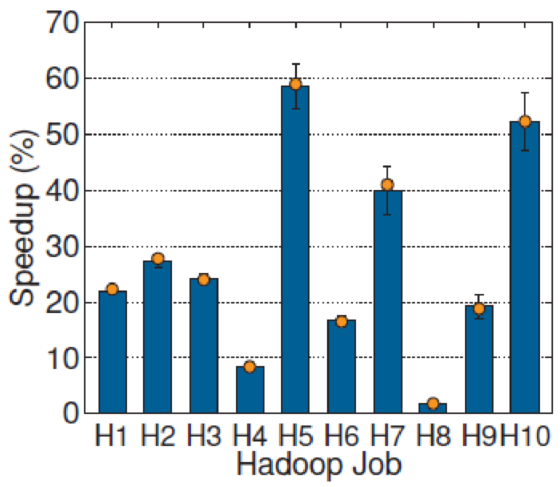


Figure showing the stages of *Quasar* upon submitting a workload.

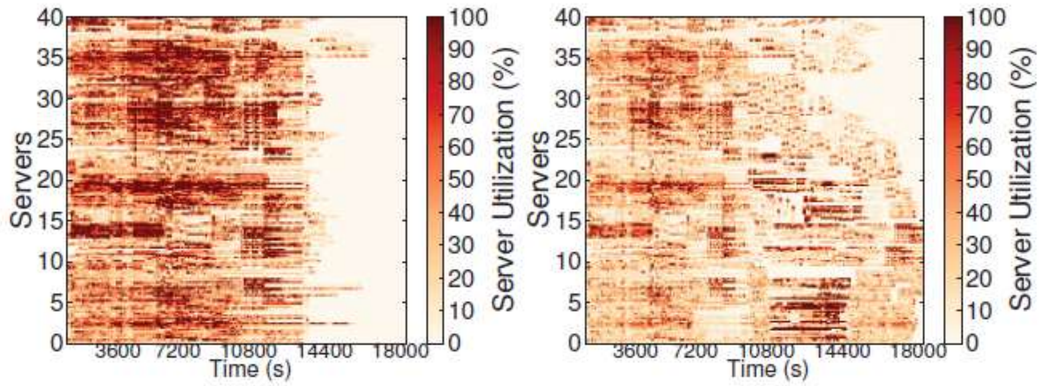
Evaluation

In the evaluation section, we compare different aspects with different frameworks showing the Quasar is best in almost every aspect. *Hadoop* is a framework that allows distributed processing of large data across clusters of computers.



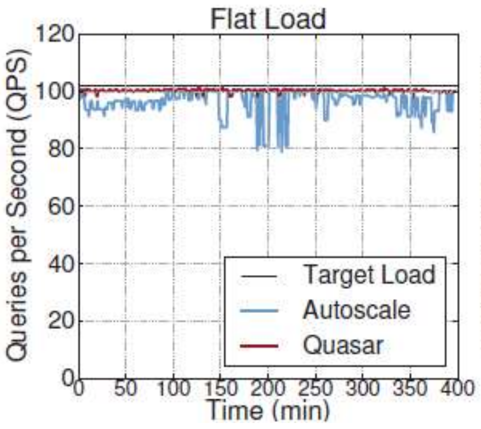
This graph shows the speedup in execution time for 10 Hadoop jobs when scheduled with Quasar instead of the Hadoop's framework scheduler. Some jobs can reach 60% speedup when launched through Quasar.

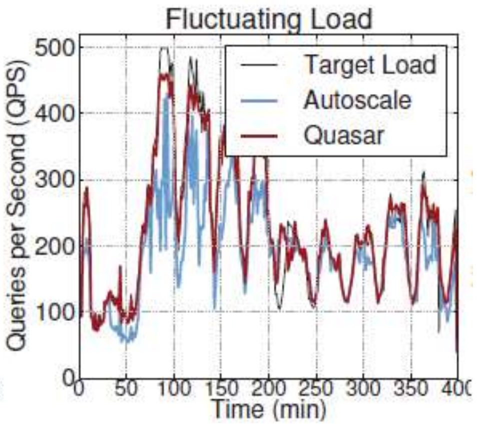
Figure 5: Performance of the ten Hadoop jobs.

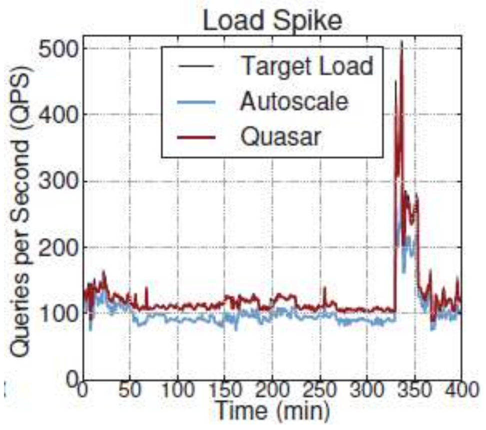


This graph is shown as a heat map, expressing Cluster utilization with Quasar (left) and other framework schedulers (right) such as Hadoop, Mahout, Spark.

We can clearly see that the left map is darker meaning more utilization sometimes reaching 100%, also a quick look on the X-axis will reveal that the execution time with Quasar was faster meaning better performance.

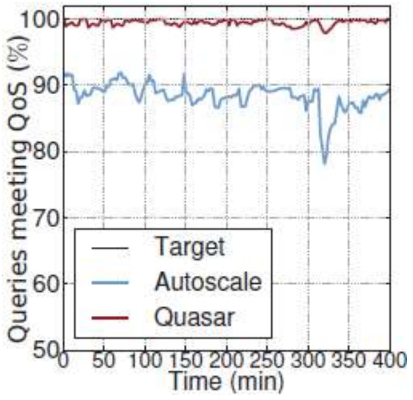




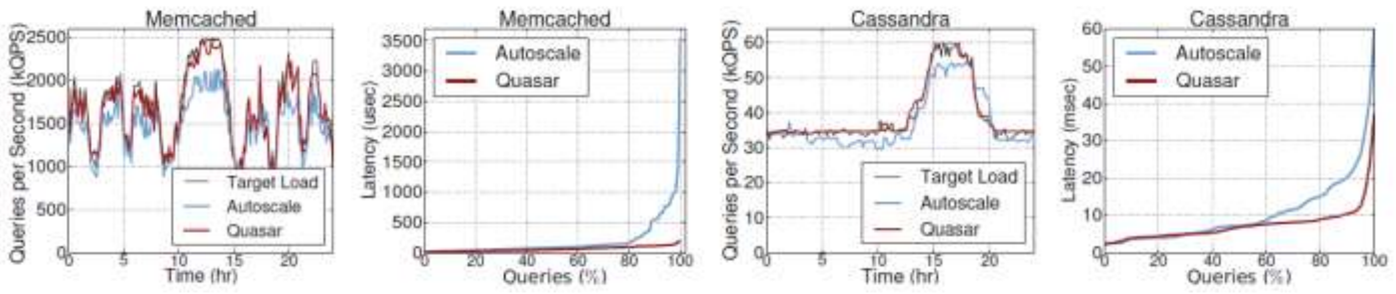


For latency-critical applications, a webserver was built on basis of the HotCRP management system and is measured by QPS(Queries-per-sec).

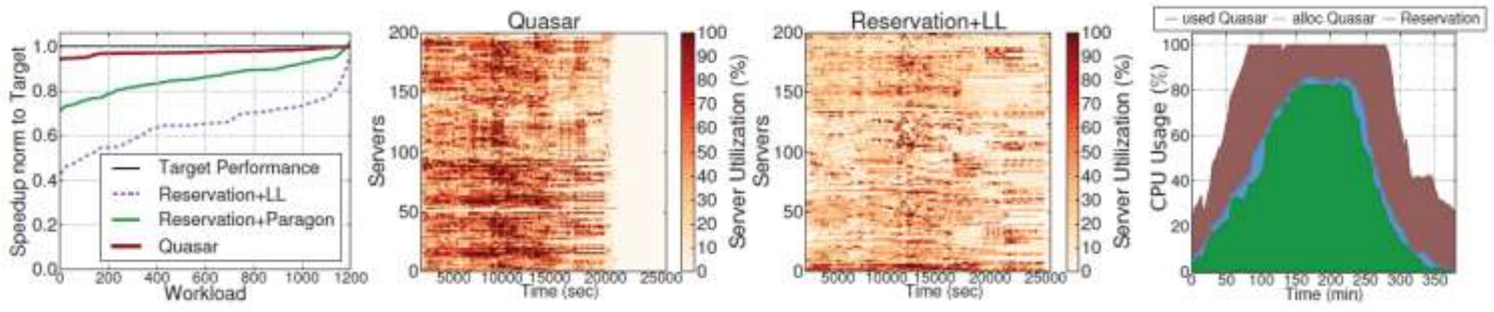
The three graphs measure the amount of QPS under different loads: Flat, fluctuating and a spiking load. In the three graphs it clearly shows Quasar managed to beat Autoscaling systems in terms of QPS and Almost preserves the target load QPS without any significant downs Meaning that Quasar is meeting the performance constraints better Than other systems.



In terms of QoS, Quasar clearly beats Autoscale systems by noticing That it almost identical to the Target graph.



Memcached is a distributed caching system, Cassandra is a distributed Database, here we compare the QPS count & latency for queries across The two systems when using Quasar or Autoscale systems.
First graph shows the count of QPS with Quasar beating those of Autoscale and Quasar almost meeting the Target load almost at every Time snap.
Graph b shows that the latency on 100% percentage of queries with Autoscale systems is huge opposed to Quasar which keeps a very low Latency. In graph d Quasar’s latency is a bit higher but it still beats Autoscale systems by far.

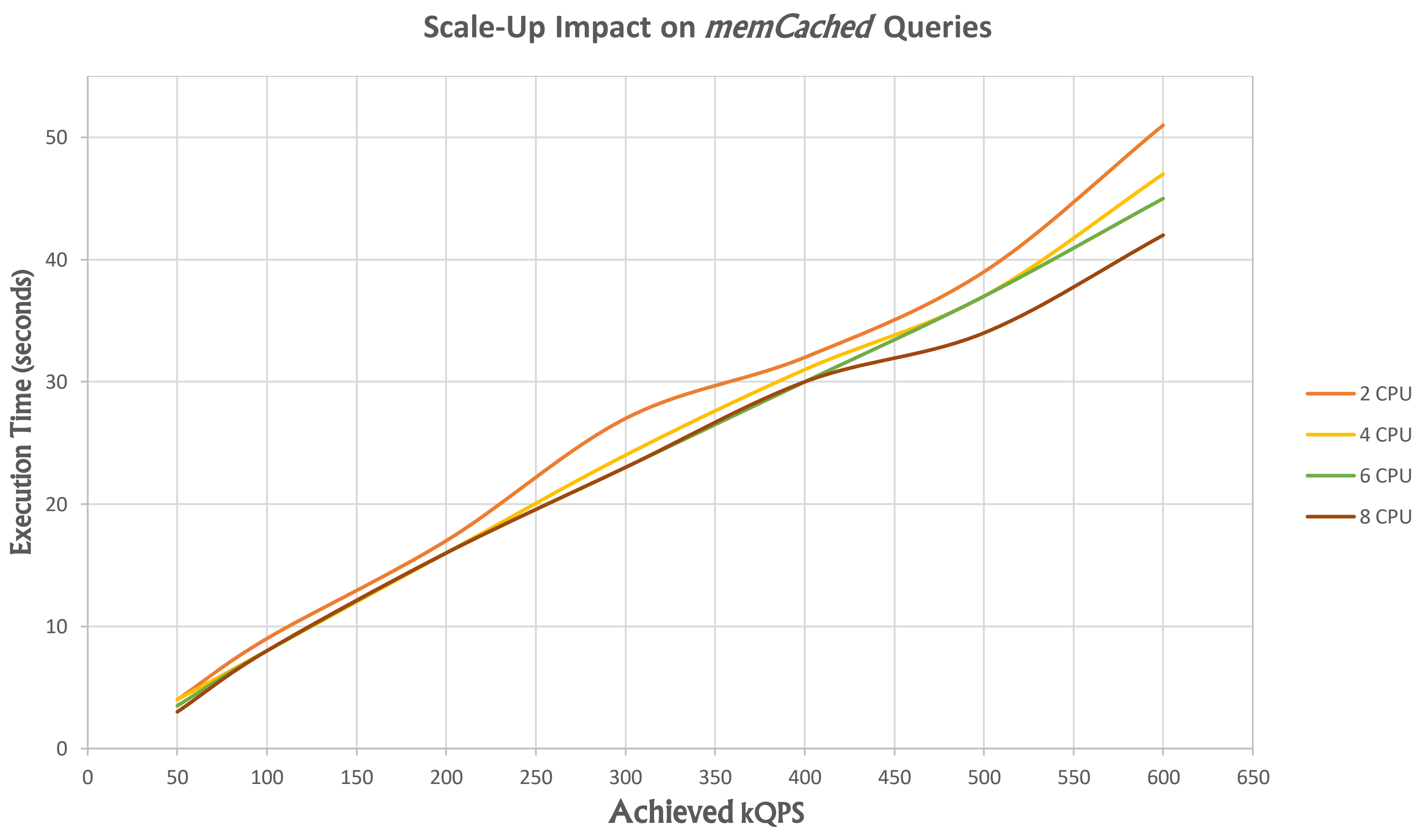


Paragon was like an early version of Quasar made by the same Researchers but it did not take into account Scale-up and Scale-out.
In graph a we see that Quasar has a bigger speedup than Paragon and Than allocating least-loaded servers.
In graphs b & c we clearly see a darker heat map on Quasar side Meaning better servers utilization plus a shorter heat map meaning Less execution time.
Graph d: in brown are the reserved resources made by the user versus The resources allocated by Quasar (blue) and showing the resources

Actually used by the cluster (green) we see that Quasar can better Estimate the amount of resources needed for the workload while Preserving cost effectiveness and achieving better utilization.

Graph Reproduction

I decided to reproduce the scale-up graph, which shows the effect of increasing the number of CPU cores on execution time/latency in queries.



The tests were done on Memcached, a distributed memory object caching system. My graph is the equivalent of graph h in the original paper in page 4, although I do not have the same resources as the original authors had (I only have 8 cores while the original authors had up to 24) the graph is similar and through analysis we can obtain the same result from the two graphs.

My test was a piece of Python code that performed a variable number of GET/SET queries (50K queries – 600K queries a different number of cores every time.

I used Pymemcache (Python module that directly supports Memcached) on an i7-7700HQ processor, lubuntu 18.04 with 8 GB of RAM. (Tests were done on my laptop)

If we compare this graph to the original one, we can easily conclude from both that Scale-Up effect is real, meaning the more cores we use the less the execution time (in the original graph the goal was to measure latency for query but I created a graph with execution time because it easier to understand). While this observation (more cores = less execution time) isn't clear at first, at the end of the graph it's clear that 8 CPUs have less execution time than 6, and 6 CPUs have less time than 4 and so on.

Unfortunately, the original dataset is not available, so I downloaded a dataset from the internet (huge CSV folder) and used it to fill Memcached system so I can perform the queries.

A link to the dataset: https://www.kaggle.com/allen-institute-for-ai/CORD-19-research-challenge

Algorithm Extension

I will propose an extension to the Quasar framework I showed earlier.

Resource Partitioning: dividing the resources between workloads on the same node/server.

Quasar does not divide the node's resources directly. Instead, it tries to allocate workloads that do not compete on resources therefore reducing interference importance.

Some of the resources that would be divided are caches, memory banks and memory bandwidth. The partitioning of resources could save us the time that is taken to evaluate and profile the workloads in term of sensitivity to interference thanks to lack of competition between workloads on resources. While this option may improve Quasar execution time we have to take into consideration the time needed to profile every workload in terms of partitioned resources, that means I'm suggesting to change interference profiling with resource-partitioning profiling combined with SVD & PQ algorithms to obtain general results as Quasar does.

On an important note, one should notice that *Resource Partitioning* is not the same as Scale-Up classification, in the first we measure the effect of cache partitioning (dynamic/static), memory bandwidth, network bandwidth and so on, and in Scale-Up Quasar only takes into account the number of CPUs, RAM and storage parameters.

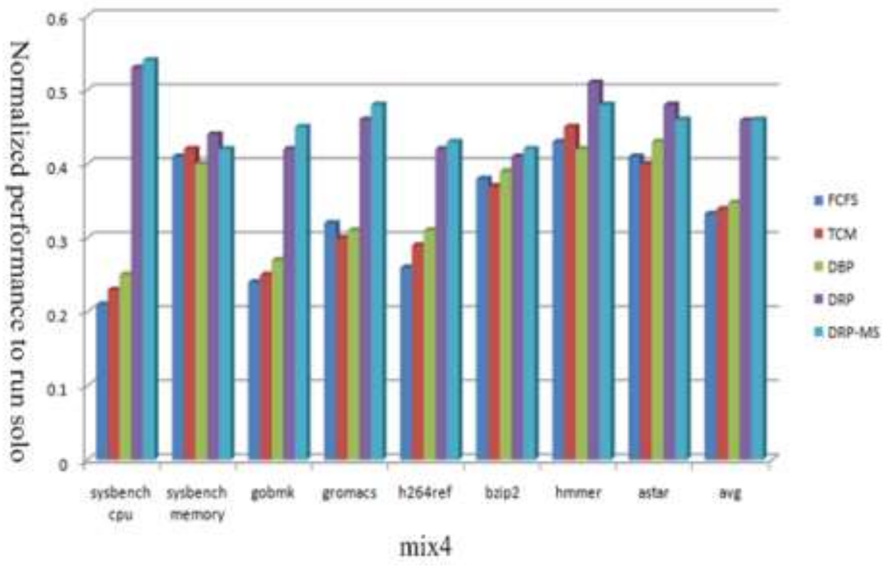
Integrating *Resource Partitioning* into Quasar should not be hard, taken into fact that the classification engine already exists and SVD & PQ reconstruction algorithms are used to provide accurate estimated results, we only have to swap interference with *resource-partitioning*.

Another suggestion I would like to add, is to integrate resource partitioning without cancelling interference classification. If We remember the way *Quasar* works it classifies workloads in terms of heterogeneity & interference and after that in terms of scale-up and last step is to scale-out (which is <u>**OPTIONAL**</u>), meaning we can add the resource-partitioning as a bonus/optional classification that could better utilize our cluster.

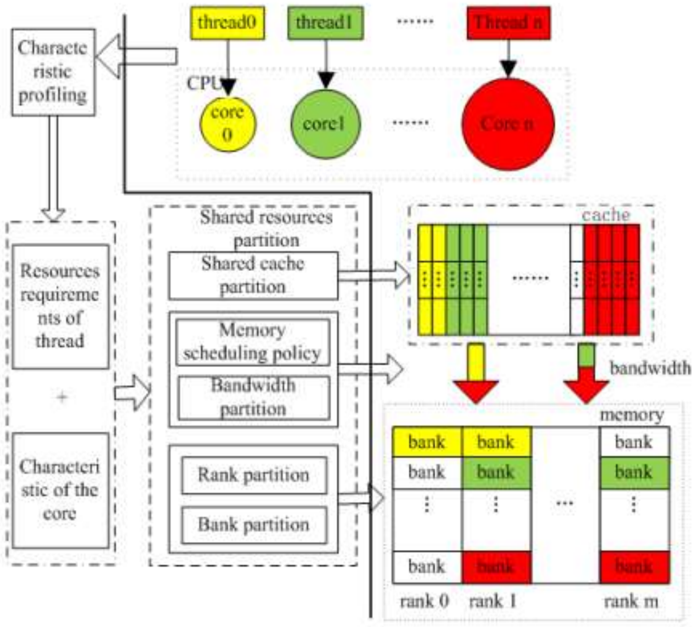
Adding *Resource Partitioning* will increase profiling time (more computations) but its benefits could be larger, and as we said before we can make it an optional classification if we want to utilize our cluster better. Some might say that this extension could not benefit *Quasar* and only produce conflicting results with the interference classification, while this might be true, after obtaining the results from both classifications, we could let *Quasar* decide which one would yield the optimal results in terms of utilization.

Resource Partitioning can be static or dynamic, meaning that resources (I'm talking about caches, network and memory bandwidth and not about cores/RAM, that would be scale-up) can be fixed until the workload finishes or that it can dynamically change according to the workload's needs.

In terms of evaluation, I am going to refer to an article about *"Dynamic Resource Partitioning for Heterogeneous Multi-Core-Based Cloud Computing"* [4], where researches introduced a dynamic resource partitioning system with stages similar to *Quasar* like profiling workloads according to memory intensity, row-buffer locality and bank level parallelism. A step further in the article, to improve fairness between workloads, the researches introduced a memory scheduling system into the DRP(dynamic resource partition system).



This figure taken from article [3], shows the improvement in performance when using DRP and DRP-MS (dynamic resources partition with memory scheduling), we can clearly see that the purple (DRP) and the light blue (DRP-MS) have the highest scores meaning better performance.



This figure shows the framework of DRP algorithm [3], the mentioned article does not use SVD & PQ methods but we can change that to obtain results faster. In terms of classifications, this figure shows the similarity between DRP and Quasar by first profiling thread's needs (the equivalent in Quasar is profiling the workload needs) and choosing the best partition to obtain maximum results.

Conclusion

We have seen *Quasar*, a QoS aware cluster manager and we have proved that the framework does work and has been proven to achieve greater results by far than existing frameworks like Hadoop and Paragon. I have learned about Cloud Computing and cluster management and how workloads are submitted and resources allocated.

We observed the shifting from resource-reservation based system to a performance-based system and the results were impressive. *Quasar* also introduced new constraints to the cloud computing area, it offers latency constraints, execution time, cost and instructions per second while most of the current systems let the users decide what resources they want.

The article also showed that most users do not know their workload needs and letting them decide will cause low utilization in most cases.

Comparing *Quasar* to *Paragon* which was made by the same researchers, has shown that scale-up & scale-out classifications will lead to more efficient resource allocations and more resources utilization at the little expense of profiling the workload a little more than *Paragon*.

Having proved that *Quasar* is an impressive framework with astonishing the researchers wrote an implementation with 6K lines of C code and the article showed that *Quasar* uses marginal resources to operate. All of these features prove that *Quasar* is a system worth taking, also it's an early concept so we know that it can be upgraded further.

Hanna Hayik

References

[1]

L. Barroso. Warehouse-scale computing: Entering the teenage decade. ISCA Keynote, SJ, June 2011.

[2]

Charles Reiss, Alexey Tumanov, Gregory Ganger, Randy Katz, and Michael Kozych. Heterogeneity and dynamicity

[3]
https://www.csl.cornell.edu/~delimitrou/papers/2014.asplos.quasar.pdf

[4]
https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7352309