Exploiting Spatio-Temporal Tradeoffs for Energy Efficient MapReduce in the Cloud

Report by Alexander Bunte, Manish Kumar

Motivation

Today more and more enterprises leverage cloud computing because of its pay-as-you-go model. One of the most popular applications in the cloud is MapReduce, a framework for processing huge datasets. However the clusters used for providing those cloud services need a lot of energy; over their lifetime costs for powering and cooling the machines are typically as big as the acquisition costs. So making the service as energy efficient as possible is a hot topic.

MapReduce in the cloud

In a cloud setting each customer gets his own cluster of virtual machines (VMs) to run his MapReduce jobs. A physical machine (PM) of the cloud hosts several VMs. Typically different types of VMs are offered by the cloud operator – for example one with two CPUs and one with four CPUs.

The paper proposes some algorithms to find an energy efficient placement of VMs on PMs with the following assumptions:

- Machines consume energy independent of utilization, so the only effective possibility for a machine to save energy is to suspend when there is no utilization.
- The runtime of all jobs is known it advance; this can for example be achieved by profiling the job runtime for a very small input dataset.

The main goal of the algorithms therefore is to minimize the machine uptime (MU – time until a PM can be suspended because of no utilization left) cumulated for all machines (CMU) in order to minimize the energy used.

Resource Wastage Metrics

The authors created some metrics that measure wastage of resources and its impact on the CMU. Resources are an abstraction of a PMs CPUs, Memory, Disks etc. They are wasted when they are not fully utilized – the less the utilization over time the higher the wastage. The metrics showed that the wastage mainly depends on spatial inefficiency (SI) and temporal imbalance (TI) and proportionally increases the CMU. SI is the amount of resources which is unused over the complete runtime of a machine, TI is the difference between the runtimes of a machines longest and shortest running job.

Algorithms

So they developed two types of algorithms, one for minimizing SI and one for TI, in order to minimize CMU. The first type is a binning algorithm that partitions the set of VMs to schedule into bins based on runtime. It is intended to optimize temporal efficiency. The second is intra-bin placement, that places the VMs per bin on the PMs such that a good spatial efficiency is achieved.

Binning can be done duration-based by assigning a distinct time interval and all VMs whose runtime fall in this interval to each bin or cardinality-based, where each bin gets a fixed number of VMs with similar runtime to avoid skewing. The parameters adjust temporal and spatial efficiency. The smaller they are the better the temporal, but the worse the spatial efficiency and vice versa. So a good trade-off has to be found.

For intra-bin placement a recipe algorithm is proposed. This precomputes all possible placements of VMs of all types on PMs and ranks them by utilization. At placement-time it chooses in each iteration the recipe matching a subset of VMs from a bin with the highest rank and schedules these VMs to a PM. Another possibility is to use a simple first-fit algorithm.

For further improvements in efficiency incremental time balancing (ITB) can be used. It exploits the scalability of MapReduce by adding more VMs to jobs, whose runtime is notable bigger than that of others scheduled to the same machines, to decrease their runtime and thus the temporal imbalance.

Evaluation

The algorithms are evaluated using a simulation framework. This generates a certain number of jobs having a deadline on execution time, a number of VMs necessary to meet the deadline and an adequate VM type. The way these parameters are assigned is configurable. The framework then simulates placement and execution of the jobs for combinations of the mentioned algorithms. Furthermore there are two hypothetical ideal algorithms to get a good a measurement on how well the real algorithms perform. The first is allowed to move VMs between PMs for free while execution of the jobs. This means it can reschedule all jobs every time one finishes and can reach nearly optimal spatial efficiency without having to trade temporal efficiency. The second just assumes power consumption of PMs to be perfectly proportional to utilization giving a lower bound for energy efficiency.

The experiments show that as expected cardinality based binning combined with recipe achieves the lowest CMU under the realistic algorithms and is only slightly worse than the hypothetical ones. The second is duration based binning + recipe. Recipe (spatial) only is already much worse and first-fit is worst. Using additionally ITB decreases CMU for all algorithms even more.

Further experiments try to find optimal values for duration and cardinality for the binning algorithms. Also the robustness of the algorithms to changes in the environment (distribution of runtimes, amount of resources on physical machines and non-uniform assignment of VM types) is explored. Here all algorithms behave similar as the hypothetical ones.

Conclusion

The results show that spatio-temporal algorithms improve utilization of a MapReduce cloud by 20-35% compared to the simple spatial ones and are robust to changes in the environment. The incremental time balancing further improves utilization by up to 15%.

Discussion

The presented approach works completely offline assuming that there is a bunch of jobs, that need to be scheduled. However in real world jobs arrive continuously and need to be executed as soon as possible to not annoy the customers. This means batching jobs until there are enough to find a good placement is typically not a good idea. Instead the scheduler could monitor remaining runtime and utilization of PMs to find a good placement for newly submitted jobs. Similarly a normal MapReduce scheduler for a non-virtualized setting, where jobs of different users run on the same cluster, could be implemented.

The approach aggregates the physical components of machines to an abstract resource amount. Because different jobs typically don't utilize all components uniformly this should better taken into account to get the best efficiency, but it heavily increases the complexity of the recipe algorithm. If the PMs are not homogeneous, the complexity increases even more. This could in the end make the scheduling very costly.

Another weakness is, that the job execution was only simulated. The authors did not run any real experiments. So it is not clear whether the spatio-temporal placement performs that well in reality or if maybe some assumptions were not fully correct – for example the fixed power consumption of active machines.