Macrothink Institute™

Network Protocols and Algorithms
ISSN 1943-3581
2012, Vol. 4, No. 1

Master/Slave Assignment Optimization for High Performance Computing in an EC2 Cloud Using MPI

Florian Schatz, Sven Koschnicke, Niklas Paulsen, Manfred Schimmler

Department of Computer Science, Christian-Albrechts-Universität zu Kiel

Hermann-Rodewald-Strasse 3, Kiel (Germany)

E-mail: mail@florianschatz.de, svk@informatik.uni-kiel.de

Received: January 30, 2012 Accepted: May 16, 2012 Published: May 30, 2012

DOI: 10.5296/npa.v4i1.1290 URL: http://dx.doi.org/10.5296/npa.v4i1.1290

Abstract

For high performance computing, cloud services offer highly scaleable infrastructures on demand. Without requiring a great deal of maintenance and financial resources, which a datacenter would need, it is possible to obtain a huge set of computing instances from, e.g., the Amazon Elastic Computing Cloud (Amazon EC2). The downsides of cloud computing are that the set of computing instances is arbitrary and that communication speed varies greartly and affects the running time of algorithms.

We present a master/slave selection algorithm for the EC2 cloud service based on a benchmark to counteract this disadvantage. The benchmark measures communication speed and the variance of communication in a NxN network consisting of N cloud instances. With the results of the benchmark, we perform a master/slave selection algorithm to maximize the number of messages between master and slaves.

The results show that our proposed method greatly increases efficiency.

Keywords: Cloud Computing, Communication Benchmark, High Performance Computing, MPI, Master/Slave selection

1. Background and Introduction

High performance computing projects usually consist of a set of algorithms building a complex solution for a certain problem. Within these algorithms, there are typically small sections that consume the main fraction of the total computing time needed. These parts are typically isolated and are being run on a high performance cluster or special purpose hardware. Both of these have the disadvantage of requiring a high investment and maintenance. If the algorithms to be run do not utilize these machines continuously, this might not be reasonable. An alternative for these applications is cloud computing, in which computing instances can be acquired instantly and payment is only coupled to the time consumed. Furthermore, the idea of sharing resources and saving energy supports the green IT paradigm.

Many scientific applications are already designed to be run on cloud services, such as Amazon EC2 (Elastic Compute Cloud) [1, 2, 3, 4, 5].

From an algorithmic point of view, the execution of an algorithm on a cloud computing service has many challenges. Many possible optimizations arise because the unknown and arbitrary physical network topology is different from the fixed infrastructure being used in a cluster with almost deterministic behavior.

Different communication schemes result in different performances in terms of speed and latency. Another point is that from one request of cloud instances to the next, the set of instances changes and with this the optimal communication pattern changes as well.

Unlike research based on selecting performance-effecient master/slave selection [6] in the sense of computational performance, we have the main objective of selecting a master based on the communication speed because this is, from our point of view, one of the most important challenges in a cloud environment as compared to conventional cluster architectures.

So far, abstraction layers like MPI (Message Passing Interface) do not offer automatic or semi-automatic optimization for communication schemes in cloud environments.

As this is very important for all communication-intensive algorithms, this research begins with this open field of research. In this work, we advance from our previous work. We showed in [4] that our benchmark for the Amazon EC2 cloud computing service helps to manage the distribution of tasks to cloud instances in order to obtain an optimal distribution regarding network communication speeds. This was achieved by measuring the average communication speeds, the variance, and the min/max value and offering the user these values along with a graphical display to assign the instances accordingly.

Starting from this, we developed a benchmark for master/slave algorithms so that the crucial task of selecting the right master in the set of cloud instances can be accomplished optimally. This is achieved by running a benchmark with user-defined parameters concerning the requirements of the application, such as message size, message count, latency, and instance count. From this benchmark, the optimal master/slave distribution is deducted. This is especially important for master/slave algorithms because in these algorithms, one master

must be tightly connected to all slave nodes, which is in contrast to distributed algorithms, in which each node works on its own. Another important point is that in normal cluster structures, the network speed is known and the selection of a master is arbitrary because all instances have an identical communication speed. In this infrastructure, traffic from a master can even be given priority by network-routers, which is not possible in cloud networks.

In the following, we will first describe the tools and methods used to measure communication speed and select optimal nodes for a master/slave-communication scheme. After that, we describe some benchmarks, applied to validate effectiveness of the chosen methods. The paper concludes with a discussion of the results.

2. Methods

Many complications can arise if one works on an unknown physical infrastructure. In the case of Amazon EC2, a set of spot instances (instances requested for a price with a maximum bid), for example, can be distributed over different data centers. This results in different physical locations and thus varying communication latencies.

The possibility of instances running in virtual machines on the same physical machine is another adverse factor. Another important problem is noisy neighbors, instances within the same routing level that are communicating extensively and reducing the amount of the shared resources that can be used. Furthermore, the maximum physical distance between any two instances increases with the number of instances because physical space limits the number of instances, servers, or racks, which also affects latency.

To capture any configuration of instances, we created a tool that measures the maximum communication speeds as well as maximum throughput for larger data sets. This benchmark is based on [4], but has been adapted and extended to the requirements of the master/slave scenario. MPI is used as a tool for the communication.

After this NxN performance measurements are completed, which represent the speeds between any two instances, a master is selected based on user-defined customizable criteria. Note: the NxN measurement takes user requirements, such as preferred message size, for the master/slave application as well.

In the following section, we explain the test of communication speed and its implementation in short, followed by the algorithm for the master/slave selection.

2.1 Testing Communication Speeds

As the speed of communication is time dependent in a cloud environment, we run the speed benchmark for a duration of about two minutes in order to get an fair estimation. We found this duration to be valid and reasonable, because the booting time for instances is in the same order of magnitude. It could be adapted to different time requirements if needed.

The communication itself is measured by the following procedure. Each instance i sends

a message to another instance j, which is randomly chosen from N-1 to counter systematic routing optimizations and mimic any communication scheme without a fixed pattern. After the message has been received by j, it is being returned to i, and the transmission time is measured. This procedure is repeated for the above-mentioned duration.

Then, the data is aggregated and output as mean values $\lceil_{\iota,\varphi}$ and standard deviation $\int_{\iota,\varphi}$ for all pairs of instances.

2.2 Implementation of Speed Tests

Speed tests are executed by a deploy script run on a client computer, which requests nodes, initializes them with the needed programs written in C using MPI, and starts these programs. For the detailed procedure, see [4].

The result of the speed tests run on N nodes are four NxN matrices.

$$S_{avg} = (s_{avg,\,i,\,j})_{i\,=\,1\,\ldots\,N,\,j\,=\,1\,\ldots\,N}\ , \tag{1}$$

$$S_{min} = (s_{min,i,\,j})_{i\,=\,1\,\ldots\,N,\,j\,=\,1\,\ldots\,N}\ , \tag{2}$$

$$S_{max} = (s_{max,i,\,j})_{i\,=\,1\,\ldots\,N,\,j\,=\,1\,\ldots\,N}\ , \tag{3}$$

and

$$D = (d_{i,\,j})_{i\,=\,1\,\ldots\,N,\,j\,=\,1\,\ldots\,N} \tag{4}$$

which represent the average speed S_{avg}, minimum speed S_{min}, maximum speed S_{max}, and standard deviation D, respectively. Therefore, $s_{min,i,\,j}$ represents the minimum measured speed between nodes i and j. For i = j, the value is not relevant and is defined as 0.

2.3 Algorithm for Node-Selection

To select the node that is best suited to act as master, a scoring function must be introduced. First, we define

$$\bar{s}_{avgi,j} = \frac{s_{avgi,j}}{max(S_{avg})} \quad (5)$$

$$S_{avg} = (\bar{s}_{avgi,j}) \quad (6)$$

$$\bar{s}_{mini,j} = \frac{s_{mini,j}}{max(S_{min})} \quad (7)$$

$$S_{min} = (\bar{s}_{mini,j}) \quad (8)$$

$$\bar{s}_{maxi,j} = \frac{s_{maxi,j}}{max(S_{max})} \quad (9)$$

$$S_{max} = (\bar{s}_{maxi,j}) \quad (10)$$

$$\bar{d}_{i,j} = \frac{d_{i,j}}{max(D)} \quad (11)$$

$$\bar{D} = (\bar{d}_{i,j}) \quad (12)$$

as matrices with normalized results. The scoring function for a node i is defined as

$$c_i = \sum_{j \in \{1..N\} \setminus \{i\}} \left(w_{avg} (\bar{s}_{avgi,j})^{n_{avg}} + w_{min} (\bar{s}_{mini,j})^{n_{min}} + w_{max} (\bar{s}_{maxi,j})^{n_{max}} + w_{dev} (\bar{d}_{i,j})^{n_{dev}} \right) \quad (13)$$

The parameters w_{avg}, w_{min}, w_{max}, and w_{dev} allow a weighting of the different measures. Each node receives a score defined as the sum of the scores for connections to all other nodes. The different measured values are raised to the powers n_{avg}, n_{min}, n_{max}, and n_{dev}, respectively, to take greater account of large deviations.

Then, the node with the best (smallest) c is selected as master.

2.4 Benchmark Master/Slave-Communication

To test the quality of the master selection, we created a benchmark simulating the communication scheme of a master/slave algorithm. An instance selected as master sends a message of size s to every other instance (the slaves). The slaves then send another message of size s back to the master. This is repeated r times, and the overall runtime is measured.

The runtime of the benchmark with the instance selected as master by the node-selection algorithm can be compared to the arithmetic average over runtimes of the benchmark with any other instance selected as master.

2.5 Appling Performance Tests

The tests are run on the Amazon EC2 service. Because the computing power should only be used when it is actually needed for an algorithm to run, so a new set of nodes is requested on each run and physical location and actual performance differs between runs.

Amazon offers different node types, different availability zones, and two types of provisions. All this influences the communication performance between nodes.

Results are written to a hard disk and extracted from the nodes after benchmark completion. For a better examination of results, it is possible to create a graphical representation of communication speeds in the form of a graph in which nodes represent instances and the lengths of the edges between the nodes represent the communication speeds (approximated, because of planarity), where a shorter edge indicates a higher speed.

3 Results and Discussion

We tested different variations of availability zones and provisioning, always using the smallest node type available. Amazon offers node types with better network performance, which should improve the overall communication speed, but the variation in speed depending on availability zones and provision type still exists.

The first evaluation was made on 20 nodes in the same availability zone. This should be the best connectivity between nodes because all instances should run in the same datacenter. After that, instances were requested from 4 different availability zones, which means that the nodes may be distributed between different datacenters and have an overall higher latency. The last test was made with spot instances. These instances are assigned by Amazon in terms of when they may be used for a price offered by the user, and they have the largest variance in communication speeds.

We used the values to configure the node-selection algorithm as shown in table 1. See section 2.3 for an explanation how they affect the node selection.

Table 1. Parameter values for node selection algorithm.

Parameter	Value	Parameter	Value
n_{avg}	1.2	w_{avg}	1.0
n_{min}	1.0	w_{min}	0.3
n_{max}	0.8	w_{max}	0.1
n_{dev}	1.3	w_{dev}	0.5

3.1 Instances from one Availability Zone

The most common use case is probably the one requesting EC2 instances from one availability zone to perform a computational task. One would expect a set of instances with an almost equal interconnection speed. Our results show that instances from one zone can have significantly varying communication speeds.

As Fig, 1 and 2 show, connectivities between nodes, even in the same availability zone, differ. Having an algorithm that uses node 18 as an master might significantly affect the total runtime.

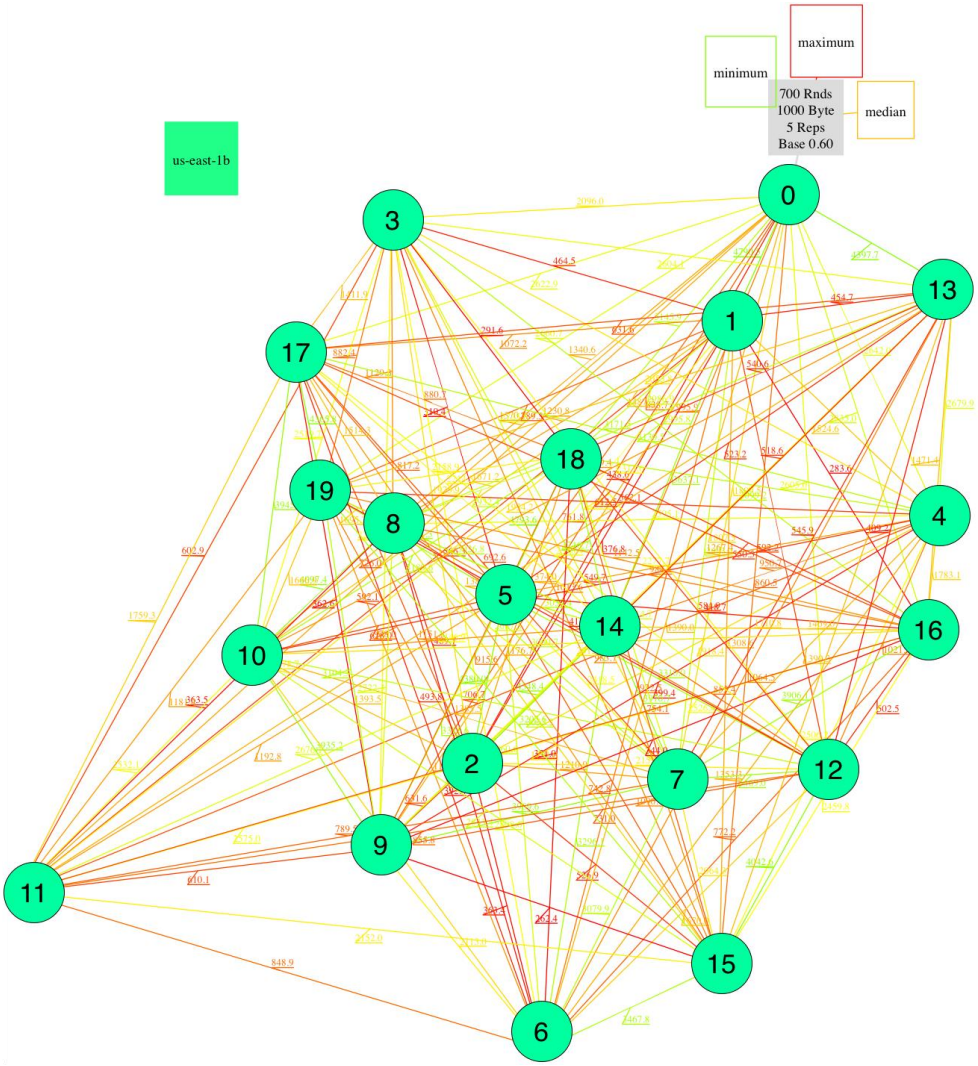

Figure 1. Graphic display of 20 instances from one availability zone. A node represents an instance. An edge between two nodes has a length that relates to the latency measured by the performance test. Clusters of instances with fast interconnections can be easily identified by the user.

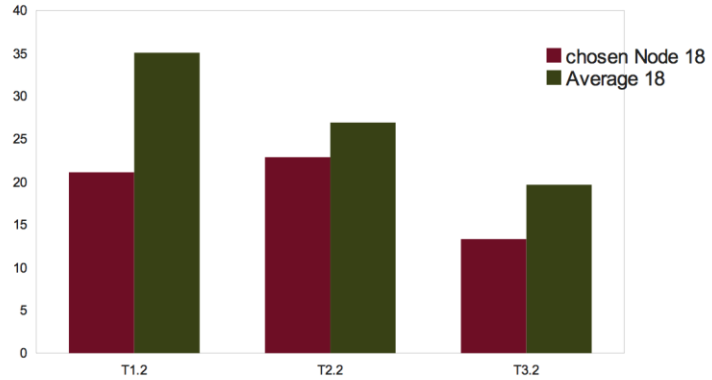

Figure 2. Speed chart of 20 instances requested from EC2 within one zone. Comparing the overall runtime in seconds (y-axis) of the test with the node suggested as master by the algorithm and the median of runtimes when any other node is used as master. Message size was 1000 bytes in the first test (T1.2), 100 bytes in the second test (T2.2) and 10 kilobytes in the third test (T3.2). Tests were repeated 40 times each.

It is often the case that one or two instances have exceptionally bad connectivity to all other instances. The overall performance can be increased when more instances than needed are requested and those with the worst connectivity are discarded before beginning the calculation. We integrated this functionality into our tool. The worst two instances are discarded just after the initial speed tests. Fig. 3 shows that after that, another master is chosen and the overall performance is increased.

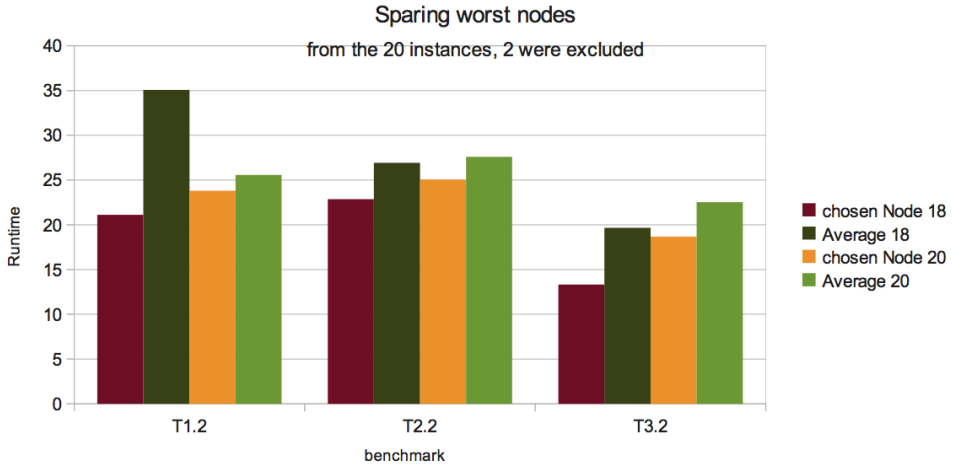

Figure 3. Speed chart of 20 instances requested from EC2 within one zone. Comparing performance with and without discarding the two instances with the worst connectivity. Runtime is given in seconds.

3.2 Normal Instances from Different Zones

When requesting instances from EC2, one can select the zone of availability required. Amazon offers different availability zones, which can be used for instances that are independent from one another. If instances from one zone are required, e.g., us-east-1a, instances from this zone are created. One might request instances from different zones in order to increase reliability by using independent parts of a data center.

Our previous tests confirm that interconnections between instances from one zone are significantly faster than instances from different zones. Selecting a good master becomes more important with the introduction of multiple availability zones.

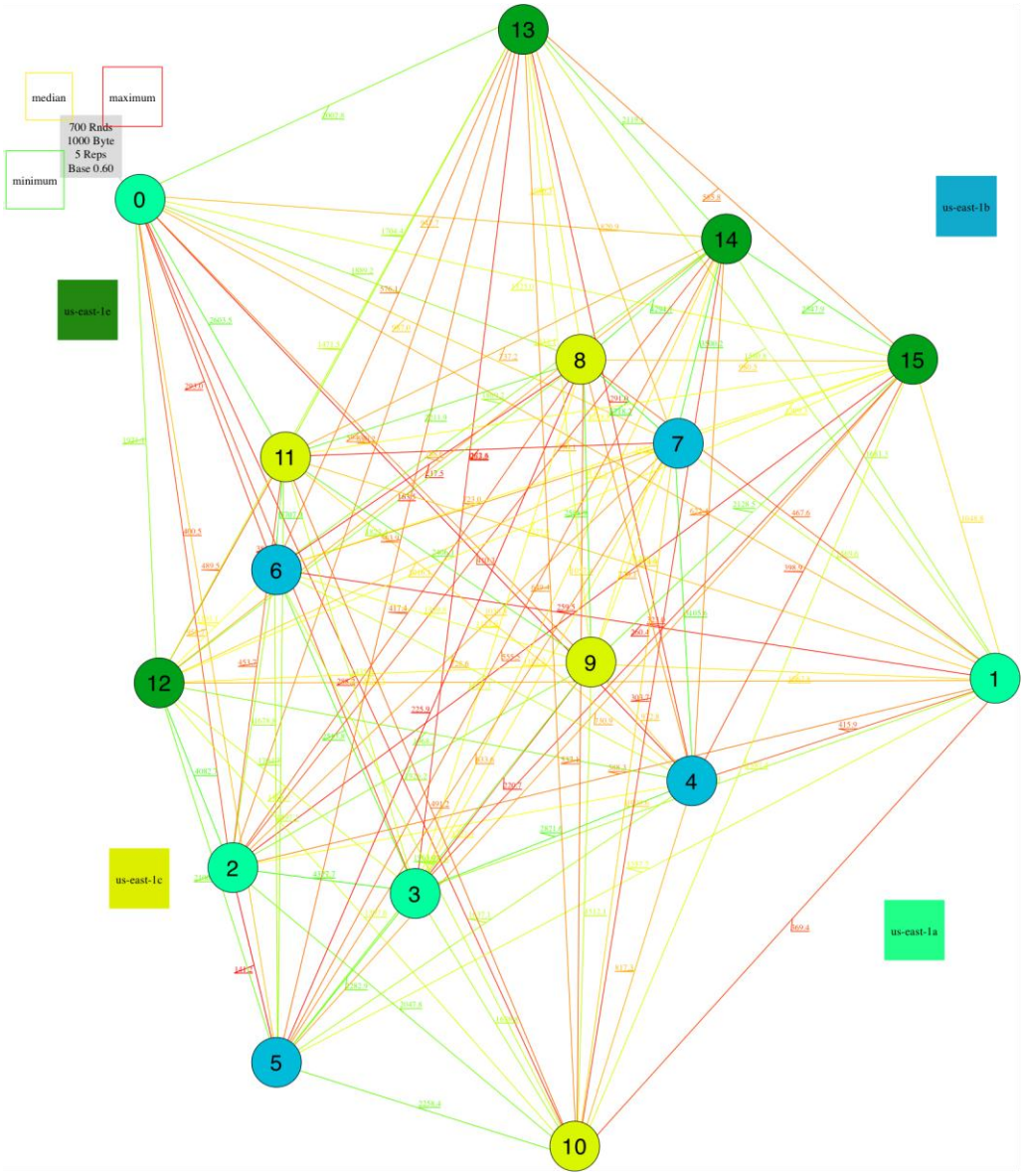

Figure 4. Graphic display of 16 normal instances requested from EC2 from 4 different availability zones. The chosen zones were us-east-1a, us-east-1b, us-east-1c, and us-east-1e. The color of the node shows to which zone the represented instance belongs.

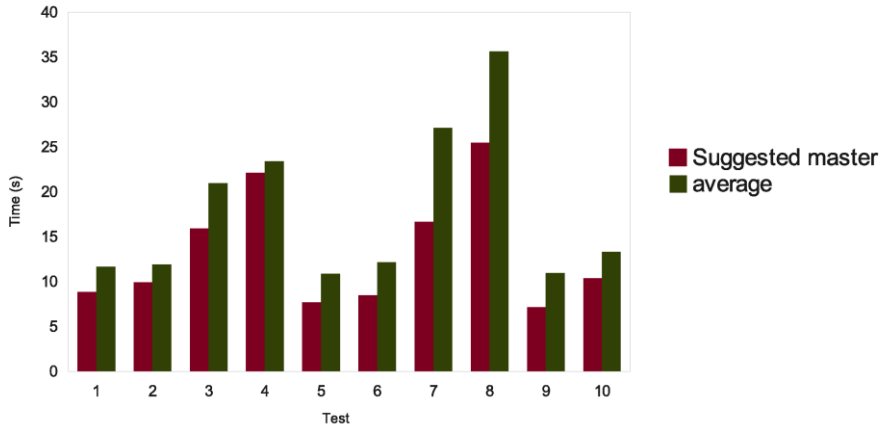

Figure 5. 16 instances requested from EC2 from 4 zones. Comparing the total runtime when the suggested master is used against the mean of the runtimes with any other node as master. Tests were made with different message sizes. Two successive tests use the same message size. The first two tests were made with 1000 bytes message size, then 100 bytes, then 10000 bytes, then again 100 bytes and finally 10000 bytes again.

As shown in Fig. 5, the performance can be increased significantly by choosing the master our algorithm suggests. Even with high variations in performance between tests, runtime when choosing the right master is always better than the average runtime.

3.3 Spot Instances

Spot instances are EC2 instances that are requested without a timely constraint, but with a certain limit in terms of cost per hour. These instances are then made available as soon as the demand and bids compared with the request decrease. The requested instances might be in different zones.

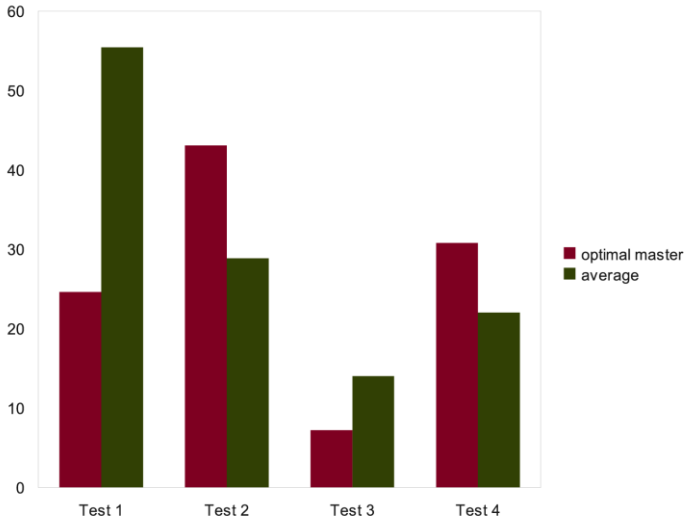

Figure 6. 25 spot-instances requested from EC2. All instances could be obtained from the same availability zone (us-east-1a). Comparing the total runtime (in seconds) when the suggested master is used against the mean of the runtimes with any other node as master. First test used 25 kilobytes messages size, second test 100 bytes, third test 10 kilobytes and forth uses a message size of 1 kilobyte. Only in tests with bigger message size the results for the suggested master are superior to the average runtime.

In our tests, we received 25 instances from one availability zone. As shown in Fig. 6, in the first benchmark, the selection of the master suggested by our algorithm increased the performance dramatically, by over 50%. Repeating the tests shows a high variation in performance. In the second and fourth tests, the performance with the suggested master is actually below average. This shows the difficulty of conducting network performance analysis for cloud systems. In the second and forth tests, very small message sizes were used (100 bytes and 1 kilobyte). There seems to be higher variation of performance for small message sizes.

4 Discussion and Conclusions

In this work, a master/slave selection algorithm for Amazon EC2 has been presented. It measures the communication speeds between instances and proposes an optimal master based on the current instance set.

Test results show how the algorithm affects the master/slave selection compared to a arbitrary choice of master, tested with nodes from one or multiple availability zones.

The tools created in this work can be used to find the optimal master for any master/slave algorithm before running it on Amazon EC2. Due to the high volatility of the communication speed between instances of the Amazon Cloud, an automized optimization can contribute to increased execution speed of algorithms running in the cloud.

5 Availability

The implementation is publicly available at this url:

http://www.informatik.uni-kiel.de/~fsch/cloud-masterslave/

The password for the .zip file is: fschsvk

6 Future Work

Further work will include a MPI and EC2 independent implementation in C. This enables usage of the communication optimization in settings where MPI is not used. The tools may then be included as library to applications written in any language which supports calling of external C-functions. While the actual benchmark-implementation is already cloud-

independent, deploy scripts need to be expanded to support more cloud-providers.

Other work will be support for benchmarking and optimization of different communication schemes like ring or hypercube.

References

[1]. Constantinos Evangelinos, Pierre F. J. Lermusiaux, Jinshan Xu, Patrick J. Ha- ley, and Chris N. Hill. Many task computing for multidisciplinary ocean sciences: real-time uncertainty prediction and data assimilation. In Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers, MTAGS '09, pages 14:1–14:10, New York, NY, USA, 2009. ACM. http://dx.doi.org/10.1145/1646468.1646482

[2] B.T. Langmead. Highly Scalable Short Read Alignment with the Burrows-Wheeler Transform and Cloud Computing. PhD thesis, 2009.

[3] Sangmi Lee Pallickara, Marlon Pierce, Qunfeng Dong, and Chinhua Kong. Enabling large scale scientific computations for expressed sequence tag sequencing over grid and cloud computing clusters. In PPAM 2009 Eigth International Conference on Parallel Processing and Applied Mathematics, pages 13–16, 2009.

[4] Florian Schatz, Sven Koschnicke, Niklas Paulsen, Christoph Starke, and Manfred Schimmler. Mpi performance analysis of amazon ec2 cloud services for high perfor- mance computing. In Ajith Abraham, Jaime Lloret Mauri, John F. Buford, Junichi Suzuki, and Sabu M. Thampi, editors, Advances in Computing and Communications, volume 190 of Communications in Computer and Information Science, pages 371–381. Springer Berlin Heidelberg, 2011.

[5] Michael C Schatz. CloudBurst: highly sensitive read mapping with MapReduce. Bioinformatics (Oxford, England), 25(11):1363–9, June 2009.

[6] Gary Shao, Francine Berman, and Rich Wolski. Master/slave computing on the grid. In Proceedings of the 9th Heterogeneous Computing Workshop, HCW '00, pages 3–, Washington, DC, USA, 2000.

Copyright Disclaimer