Resource discovery over P2P networks for the creation of dynamic virtual clusters

Carlos Paulo Ferreira Santos
P2P-Clusters: criação dinâmica de clusters em cycle-sharing

Instituto Superior Técnico
CPFSantos@netcabo.pt

Abstract. The importance of cycle-sharing and distributed computing has grown in the past years and the amount of internet traffic over peer-to-peer networks is increasing. The methods that peer-to-peer applications use to maintain scalability and perform their goals can be used to improve upon current grid systems to facilitate the creation and maintenance of dynamic virtual clusters allowing them to grow further or perform better. We intend to draw upon the strengths of P2P and grid systems to create a system capable of dynamically creating or adjusting virtual clusters for the execution of distributed applications, thus allowing for higher cluster availability as well as lessen the wasted computational resources in the form of idle machines.

Keywords: Peer-to-peer, Grid computing, Virtual clusters, Cycle-sharing

1. Introduction

In the ever-evolving modern world, one of the few constants in the computer industry has been growth. This growth manifests itself both in terms of computationally more demanding applications as well as a higher availability of computational resources.

One would think these patterns of growth would neatly overlap, yet that is not so. A high amount of computer cycles are being wasted at all times all over the world, and yet, at the same time there are applications that have a strict demand for computational power that requires more than a simple computer's effort to satisfy.

Over the length of the past 20 or so years, several approaches to this problem have been created. Initially the only solution to most heavy computational processes were to have them handled by supercomputers and mainframes, but this solution was both expensive and inherently generated inefficiency.

With the passing of time the concept of a computational cluster became the norm. A group of machines, similar both in hardware and software specifications, but also connected by high-throughput and high-availability networks, dedicated to the solving of a common problem through the uses of distributed and shared computing mechanisms as one of its roles.

However, while locally capable of providing for the needs of any entity requiring such form of computational power, it still failed to provide a solution except for the most wealthy and stable of companies[3]. The common user was still without a solution, the vast resources of the internet were still not being used, and above all, there was still a high amount of computational power that was wasted, not only during idle times in computer clusters, but also in the raw untapped power of millions of personal computers and other computational devices (such as the ever popular portable cellphones, hand-held devices, etc.) all over the world connected to the internet[6].

Several partial solutions appeared with the passing of time, each focusing on a certain approach and method to solve the challenges required to implement a platform that would both maximize computational power available as well as minimize resource waste.

The concept of distributed and parallel computing itself is one of the interesting points that evolved alongside the clustering paradigm. Cycle-sharing and Grid computing are different approaches to this concept, each with their own goals and constraints. These constraints that still prevent the general public from having access to higher computational tools via the use of clusters. In a way, public computing and computer clusters are still like two islands in a sea, totally separate, due to the lack of tools to overcome such constraints. Should a tool arise that could allow for the dynamic creation and alteration of clusters based on a peer-to-peer network, we could finally reach the goal of allowing the general public to use the computational power of clusters, due to the advantage of P2P architectures.

2. Architecture

The solution presented in this paper focuses on three individual parts. WE intend to provide a more detailed look into both our system and the inherent protocol it uses to achieve its goals. In general our system will provide means for a new node to join a network in order to be able to utilize resources on remote computers as well as allow his own resources to be used by remote requests.

Network System

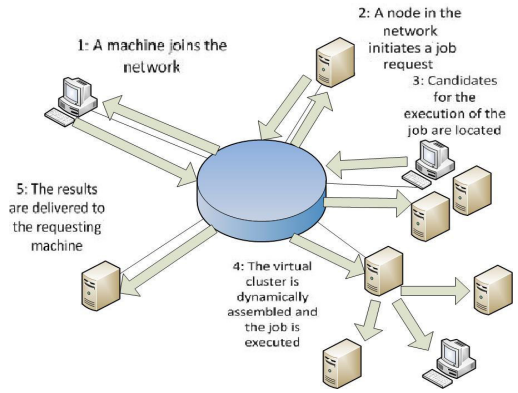

Figure 1: System overview

The figure above shows the overview of our system and a general outline of the several actions in a typical request. Our system is based on a structured peer-to-peer network of nodes. We chose this type of network to allow quick and efficient management of resources and swift routing to specific nodes, thus promoting scalability and still maintaining the decentralized architecture.

The system will allow a machine to join a network by simply knowing the IP or address of a node already in the network (as depicted in Figure 2 below), and accommodates the new node in its overlay. Both the departure of a node (loss of available resources) and the joining of one (increase in available resources) will be propagated throughout the network in a controlled way in order to not flood the system.

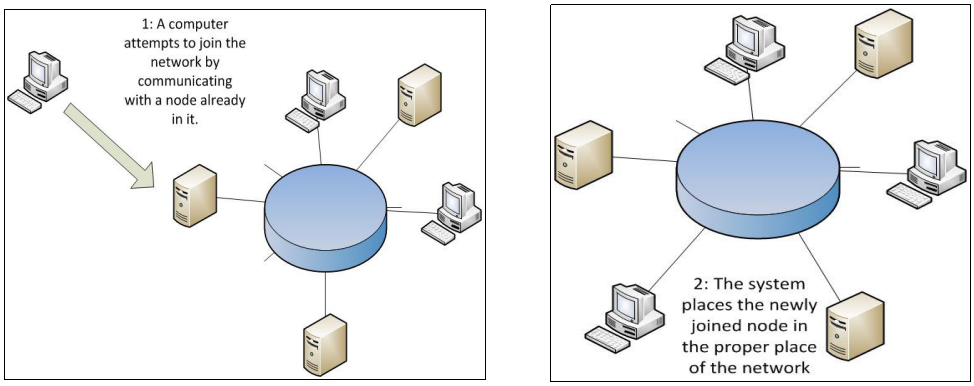

Figure 2: The entering of a node into the network.

Resource Discovery Protocol

The actual submission of a job request can be accompanied by several quality of service metrics. A request can be made for a cluster of a specific number of computers and values to each of the QoS metrics can be provided with the request to later allow our resource discovery protocol to refine the list of candidates and correctly estimate the machines that best fit the requirements of the request.

After a request is submitted, our resource protocol sets in, working on the node network maintained by our system and will attempt to locate resources that either fit the request's QoS metrics completely, or at least locate machines that fit the request (either fully or only partially), all the while maximizing their adaptation to the request's specifications. Considering the heterogeneous nature of the nodes in the network, it is most likely that only partial matches can be found[5], but the capacity for this protocol to maximize the cluster's capacities to fit the request are a crucial point. Thus, we allow each node to, upon request submission, specify minimum thresholds for both partial and global resource satisfaction.

The protocol itself is modeled in the form of PeerSim classes and its work has two steps. In the first step, the protocol must locate a list of potential candidates from within the nodes present in the network and grade them considering the metrics provided in the request. It must then select the best candidates from among that list in the second step.

The candidate gathering is carried out by examining every node in a certain neighborhood of the node in each of its axis (which represent node characteristics) that has the ideal value in three steps. The depth at which each of these four nodes tries to find viable candidates is specified in the request.

In the first step, the requesting node sends a message to each of the ideal nodes in each of the four axial characteristics. These traverse the overlay via the usual CAN method, being relayed to the neighbor node that is in the direction the message needs to take.

After this message is delivered to the nodes in whose neighborhood we will check for candidates, the second step begins. Each of these nodes will begin gathering candidates in a vicinity around it that is limited by the request's specifications. This is handled by the node looking at a number of candidates in both directions of each axis of the CAN N-dimensional space. After getting the number of candidates requested by the originating node in each of these directions, the final step begins.

This third and last step starts by compiling the list of the nodes chosen as candidates. After this list is compiled, the proper message is sent back to the originating node with the information of each potential candidate.

Cluster setup

After the machines that will be a part of the cluster are selected, our virtual appliance will then be deployed onto each of those machines. This application will be in the form of a small virtual machine capable of receiving configuration instructions as an argument, and will be also responsible for the execution of the code.

In each request, the configuration instructions will be responsible for setting up the execution of the clustered application code and contain instructions to connect to a set of machines that will be part of the cluster. After the cluster is fully deployed and set up, the execution of the job begins. Upon its completion, the node chosen as controller must collect the results and deliver them to the machine that initiated the request.

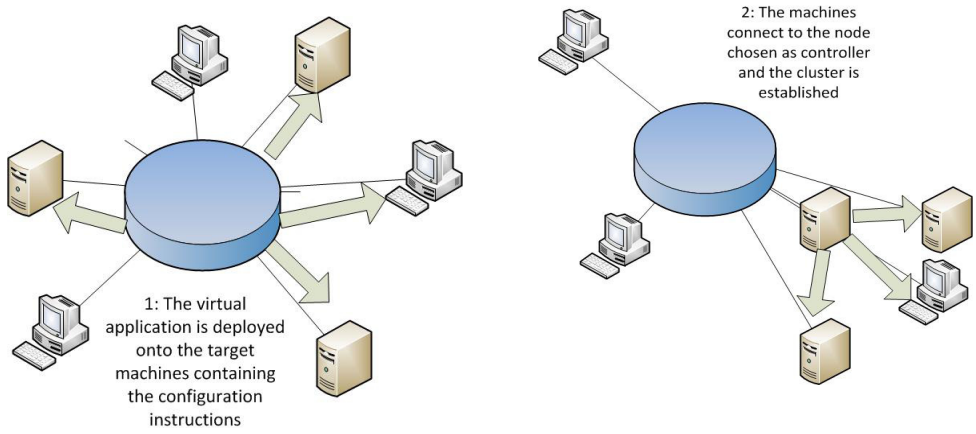

Figure 3: Virtual appliance deployment

3. Implementation

To implement our resource discovery protocol we used an approach similar to that of CAN where resources are located by obtaining from the resource key its respective set of coordinates in the N-Dimensional space. After obtaining the resource's coordinates, getting to the resource is a question of forwarding the request to the neighboring node that is in the direction of the set of coordinates.

This approach will require a strict placement of resources. In order for the location to help us, we must have to set similar machines in similar spaces, thus ensuring that should we require, for example, several machines with 3000 MHz of CPU, 2 GB of available RAM and low latency between them, we can know where in the space to look. We ensure this by making sure each node is set to handle a section of the CAN-space related to its characteristics upon joining.

In order to avoid two machines falling into the same space, there must be a certain degree of leeway in the exact coordinates. Two machines with the exact same components might end up at coordinates X,Y,Z and X+-α,Y+-β,Z+-γ. We will now provide a simple depiction of the progress of the resource location protocol in this approach.

Resource discovery

-Locate the coordinates of a resource from its characteristics and key;
-From the local node's coordinates, locate the neighboring node that lies in the direction of the coordinates of the resource we obtained in the previous step;
- If the resource is located in that node's space, stop, if not, we locate the next neighboring node that is in the direction of the coordinates;
- We then repeat the previous two steps

Cluster Deployment

To facilitate cluster setup we will adhere to the Open Virtualization Format (OVF) standard. This will allow us to utilize .OVA files that can be customized from preexisting templates in order to allow us to refer our virtual appliance to resources located on virtual disks containing the application to be executed. These packages can be extended with configuration data to be accessed by the virtual machine post-deployment.

Messages used

In our system there are several types of messages, each with their own structure. The first one encompasses the Join and Leave messages. Within this message we find the identifiers of both origin and target node, as well as the type definition and, for simulation purposes, the counter for the number of hops, which would not be present in the final system.

The second type of message is the one used in the candidate gathering process. It contains the same fields as the above message, with the additional information for the maximum depth at which we will look for candidates around each of the four “optimal” nodes.

The next type of message is the one seen when the candidate gathering process is over and we have the list, to be sent back towards the origin node. Its fields are those present in a join message, with the addition of the list of nodes selected as candidates.

The fourth message type is the one used in the cluster forming process. It contains the basic four fields of the Join message, with the additional fields of the list of nodes selected as members of the cluster, plus the payload of the OVF file that will initiate the cluster's virtual machines.

The final message type is employed after, and as a reply for a successful join request. It contains much more information than the previous messages. It contains the identifiers of each of the immediate neighbor nodes, plus the boundaries of space assigned to this node. The message counter is also present for simulation and evaluation purposes.

4. Evaluation

To evaluate our application, we need to demonstrate that it correctly satisfies its requirements, and in an efficient manner, both in terms of achieving its goals and in doing it as effectively as possible. We begin by analyzing the system's behavior in simulated scenarios (as realistically as possible in a large population) , and show that the results match the expected outcome for the proposed architecture as a whole, as well as for each of its components and message protocols.

Join and leave

In the first test, we measured the number of messages taken for any node to join the network overlay when the number of nodes is maintained, for several sizes of the overlay. As we can see in Figure 4, the distribution of message counts fits very similar to a logarithmic expansion, which is coherent with the expected outcome of $\Omega (n^{1/d})$ where n is the number of nodes present in the overlay and d the number of characteristic axis used, which in our case is 4. The leave process is analogous to the join process. The node in question sends a message to a known node, which then relays it to the neighbor nodes until it reaches the nodes that will then be involved in the partition (or merge) action.

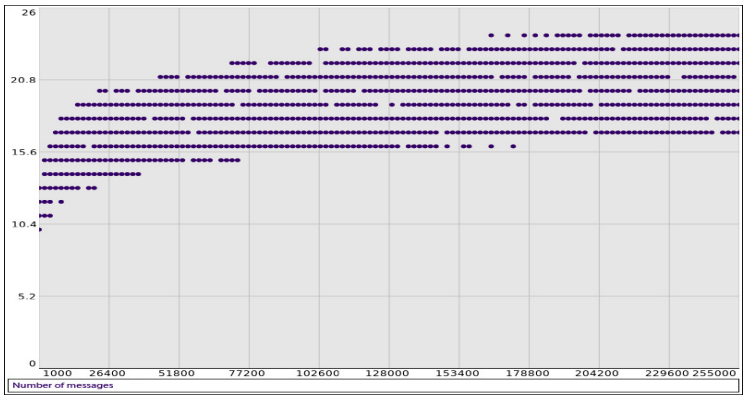

Figure 4: Number of messages taken for a Join message

X axis: Size of the network overlay, Y axis: Number of messages taken

However, in reality networks are not as stable, specially in the case of the internet, which is the main network we aim our architecture for. To more correctly estimate the system's behavior in a network where nodes can (and will) join and leave in a way we cannot predict, a test was prepared in a network with nearly constant joins and leaves. For the vast majority of of cases, the behavior will be exactly equal to the previous cases. However, in the rare case that one of the nodes in the path the message should travel is disconnected from the network after the message is emitted and before it reaches that point in the path (or the destination node), the message will have to be resent. The message first takes the path back to the origin node, starting from the node that detected the leaving node, and the origin node will again transmit the message.

Candidate gathering

The process for finding candidates that fit the specifications of the request requires the exchange of more messages. In general, it can be divided into three independent phases.

First, the node generating the request will send messages to the ideal node in each of the axis representing the four characteristics present in the system. This increases the number of messages by a factor of four, but also ensures a larger diversity among nodes selected as candidates.

After each of these nodes receives the message, the second phase consists in starting a small controlled flooding routine to each of its neighbor nodes, repeating the process until each node has obtained a certain number of candidates (which is determined in the configuration file, for example, 32 candidates).

In the final phase, each of the four nodes then sends a message to the requesting node containing the candidate lists found independently. The first and last process are quite similar to the handling of a join or leave message. The collection of nodes requires a fixed number of messages, so it can be considered a constant value. This would lead us to conclude that the sum of these processes is O ($n^{1/d}$), where d is the number of axis, which in our case is four, which once again is the expected result, in line with common scalability criteria.

In Figure 5 we can see the distribution of messages in the candidate gathering process for a candidate list depth of 4.

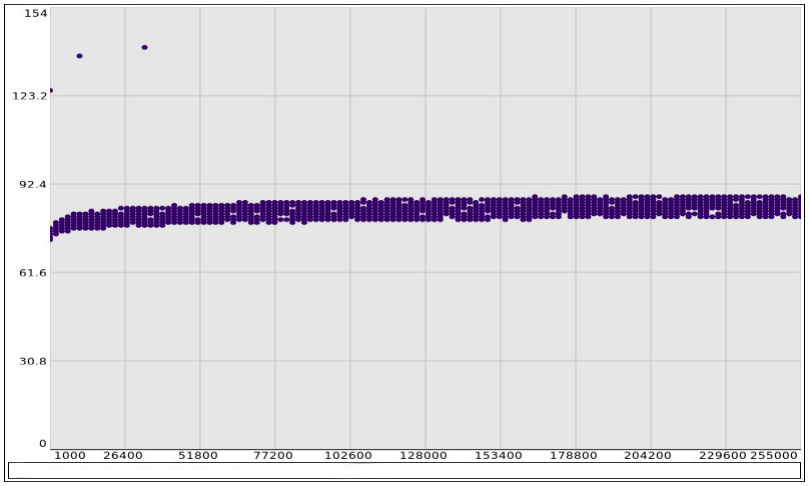

Figure 5: Number of messages taken for a candidate gathering message

X axis: Size of the network overlay, Y axis: Number of messages taken

Dynamic cluster forming

The final type of message interaction implemented by our architecture is the cluster creation message which occurs after the nodes which will be part of the virtual cluster (either as controller or regular cluster members) are selected. This message is similar to a join or leave, but only slightly larger in size. Since we are only concerned with the number of messages exchanged for now, we will focus on that aspect and examine each message's size in the following subsections.

Figure 6 displays the average number of messages required in the final phase to establish a cluster. In this case, we used candidate depth = 4 and cluster size = 4 (3 slave + 1 controller).

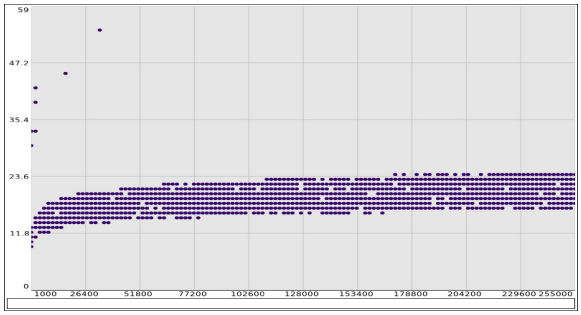

Figure 6: Number of messages taken for a cluster forming message
X axis: Size of the network overlay, Y axis: Number of messages taken

Size of the local data

We must also determine how much space is required of each node in order to maintain the overlay's integrity and the system's functionality. With our architecture, we require only that each node knows the location of each neighboring node in each axis, plus the space the current node is assigned to (possibly an extra neighbor in each direction to speed-up recovery when the neighbor fails). This means a node needs a total of 8 variables of type INT (2 for each axis) and 8 variables of type CANNode (again, 2 for each axis). This brings the total space assigned to information about the overlay to 100 bytes.

Of course, in addition to this space, each node also needs to store its own attributes in memory, which can be done with 4 more “int” variables, or 132 bits, for a total of 932 bits or 117 bytes.

5. Related work

We have examined how our system performs in isolation, but an important part of a system's viability is how it compares to other similar systems in existence. We will perform this comparison focusing on analysis of each system's properties with the aid of some quantitative data.

Message routing

System	Type of network	Number of hops	How many nodes receive the message
Gnutella[4]	Unstructured	Variable	All connected to the super-peer

Chord[2]	Ring-like topology	*O (log N)*	Message target only
Pastry[5]	Self-correcting proximity network	*O (log N)*	Message target only
CAN[1]	N-dimension space	$O(N^{1/d})$	Message target only

Being a system based on a modified CAN, our message routing follows mostly the same rules. A direct comparison shows that systems based on chord and pastry architectures scale slightly better with the size of the network overlay. Unstructured peer-to-peer based systems will scale much worse.

In practical terms, one could expect numbers in the millions of nodes without noticeable performance issues. A network overlay of 1 million would cause our system to take in average a number of messages in the mid-thirties, which when compared to a number of around 23 for a 10000 node network means that with an increase in network size of 100 times would only lead to an increase of 52% in the average number of hops taken to deliver a message.

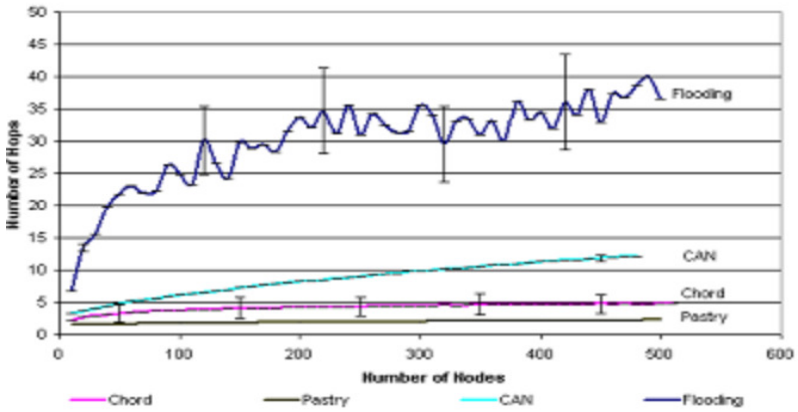

Figure 19: Comparison of message routing in different systems

Size of Local State

In the case of Chord-based systems, a node must maintain a finger table containing m entries as node descriptors. Considering that Chord's network overlay supports a number of nodes of 2^m.[2]

From this information we realize that, unlike our system, Chord requires increasing local space of each node as the network size increases, while our system maintains its required size.

Our system needs to store exactly two neighboring nodes for each axis. Considering we use four different characteristic axes, we could consider that for any network with more than 256 nodes, our system occupies less space. For a network of 1 million nodes, Chord will require 20 node identifiers, while our system will continue using only 8.

6. Conclusion

We present in this paper a system that can overcome the obstacles preventing the common user from fully using clustering tools, a system capable of efficiently locating remote resources, setup specialized clusters and allow a high degree of control to the user, in the form of quality-of-service metrics that specify in fine-grain exactly the characteristics that such a cluster needs to best fit the application being ran. This system performs this in over a peer-to-peer overlay of nodes in order to allow for greater scalability with a growing number of nodes, as well as faster resource discovery.

Our evaluation methods, to which we subjected the system and architecture, measured its performance both in isolation as well as when compared to other systems. As expected, even though the number of messages required for each of the phases of request handling grew with the overlay size, its growth was much slower than the growth of the network itself. In fact, at 250 000 nodes in the network, we were seeing around 25 messages for a typical message routing from one node to another, meaning 0.01% of the size. As the number of nodes in the overlay increases, we will notice even higher efficiency by our system. This indeed demonstrates that our system can scale very efficiently with the overlay size, which is one of the main requirements and expectations we had about the system.

The local size, while for a smaller number of nodes present in the overlay is slightly bigger than the space required for systems like Chord or Pastry, does not grow significantly at all with the number of nodes in the overlay, which is a substantial advantage over other systems.

All points above could be considered essential for any system to be considered viable and worthy of deploying after implementation. We ensured that our system is up to the task with several evaluation measures that proved how well the protocol performs, both in scalability, speed and effectiveness in handling resources.

Acknowledgements: This work was partially supported by national funds through FCT – Fundação para a Ciência e a Tecnologia, under projects PTDC/EIA-EIA/102250/2008 and PEst-OE/EEI/LA0021/2011.

Bibliography

1: S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, A scalable content-addressable network, 2001, Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, ACM

2: I Stoica, R Morris, D Karger, and M Kaashoek, Chord: A scalable peer-to-peer lookup service for internet applications, 2001, Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, ACM

3: STIMI, K.O., CLUSTER COMPUTING, 2008, COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY

4: Gnutella , http://rakjar.de/gnufu/index.php/Main_Page

5: Rowstron, A; Druschel, P, Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems., 2001, Microsoft Research LTD

6: Silva, JN; Ferreira, P; Veiga, L, Service and resource discovery in cycle-sharing environments with a utility algebra, 2010, 2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS), IEEE

7: Nisan, N, The POPCORN Market – an Online Market for ComputationalResources, 1998,