Future Generation Computer Systems 29 (2013) 352–360

Contents lists available at SciVerse ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

High throughput computing over peer-to-peer networks

Carlos Pérez-Miguel *, Jose Miguel-Alonso, Alexander Mendiburu

Intelligent Systems Group, Department of Computer Architecture and Technology, The University of the Basque Country, UPV/EHU, Paseo Manuel de Lardizabal, 1, 20018, San Sebastian-Donostia, Spain

ARTICLE INFO

Article history:
Received 1 November 2010
Received in revised form
20 July 2011
Accepted 5 August 2011
Available online 25 August 2011

Keywords:
Peer-to-peer networks
High throughput computing
Distributed hash tables

ABSTRACT

In this work, we present a proposal to build a high throughput computing system totally based upon the Peer-to-Peer (P2P) paradigm. We discuss the general characteristics of P2P systems, with focus on P2P storage, and the expected characteristics of the HTC system: totally decentralized, not requiring permanent connection, and able to implement scheduling policies such as running jobs in a (non-strict) FCFS order. We have selected Cassandra as the supporting P2P storage system for our purposes. We discuss the basic aspects of the system implementation, and carry out some experiments designed to verify that it works as expected.

1. Introduction

Peer-to-Peer (P2P) systems, initially designed for file sharing (Napster [1], Gnutella [2]) are gaining popularity in industry and academia as a powerful mechanism to support other types of applications. Some proposals (or, in some cases, working realities) are massive file storage systems [3], name indexing [4] and voice calls over IP [5]. The computational science & engineering community is paying attention to P2P systems as the foundation of a large computational resource [6].

In this work, we focus on one particular paradigm of massive computation: High Throughput Computing (HTC). An HTC system can be defined as a platform able to execute a large number of jobs per unit of time. These jobs are *independent*, meaning that they can be submitted by different users, and that jobs submitted to the system can be executed in any order—although some sort of first-in-first-out order is usually expected.

Existing HTC systems, like Condor [7] or Boinc [8], have one important characteristic that make them potentially weak: they require a central administration point. This central point could impose limitations in system scalability and also in fault tolerance. In this work, we propose a design of an HTC system based on P2P protocols in order to overcome these limitations. In the pure peer-to-peer philosophy, all the members of the proposed system should be capable to carry out administrative tasks to maintain the system operational, in addition to executing jobs. Ideally, this P2P–HTC system should perform like a non-P2P in the absence of failures, and should scale to large networks. A prototype implementation of our proposal (which does not incorporate the full range of planned features) has been used to carry out a collection of experiments. These allow us to check that the system is operational and to assess its performance. The code of the system is available to the research community by direct request to the corresponding author.

The rest of the paper is organized as follows. Section 2 describes a few basic concepts about peer-to-peer networks. Section 3 presents some ideas about P2P-based intensive computing systems, and the minimum set of characteristics that should be included to provide this kind of service; we also discuss some works carried out in this field. In Section 4, we focus on distributed, P2P storage systems, paying special attention to a particular class (distributed hash tables) and a particular implementation, Cassandra, that have been chosen as the foundation of our HTC proposal. The proposal itself is described in Section 5. A prototype implementation of our HTC system is tested in Section 6. Finally Section 7 summarizes the main conclusions of this paper, and proposes future lines of work and research.

2. Peer-to-peer networks

Peer-to-peer systems are distributed systems in which there is neither a central control point, nor a hierarchical structure among its members. In a P2P system, all nodes in the system have the same role, and are interconnected using some kind of network (usually, Internet), defining an application-level virtual network, also called *overlay*. Nodes communicate using this overlay, in order to find information, share resources or communicate human users.

* Corresponding author. Tel.: +34 685 706 943.
E-mail addresses: carlos.perezm@ehu.es (C. Pérez-Miguel), j.miguel@ehu.es (J. Miguel-Alonso), alexander.mendiburu@ehu.es (A. Mendiburu).

0167-739X/$ – see front matter
doi:10.1016/j.future.2011.08.011

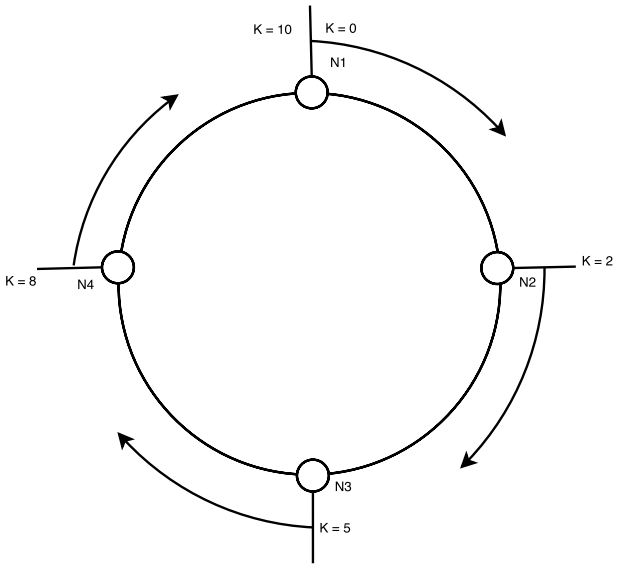

Fig. 1. Key space [0-10] mapped over a 4-node DHT.

Unlike grid systems, P2P systems do not interconnect well-defined groups using highly reliable networks. They are based on large numbers of unreliable users using unreliable network connections. Even under these limitations, they manage to provide some interesting services and features such as scalability, fault tolerance, efficient information search, highly redundant storage, persistence or anonymity.

According to [9,10], we can classify P2P systems into two basic types of *overlays*: structured or non-structured. Those of the latter type are formed by randomly connected nodes. Since there is no structure, *flooding* routing protocols are used to communicate peers. This way any network node can be reached from any other point in the system, but at the expense of an important efficiency penalty, that depends on the size of the system.

When we talk about structured overlays, we refer to systems in which there is a well defined virtual topology, and each piece of the network *content* (whatever it means, depending on the application) is stored in a well-defined network node. If we focus on data storage, we say that this type of P2P systems implement Distributed Hash Tables (DHT) [11,12], in which objects are located in a node (or nodes) chosen in a deterministic way.

Let us consider an overlay network with N nodes, each one with a different *ID*, and a (much) larger *key space*. A *hash function* provides a map of a key onto a node ID—there is a single map, in such a way that the node that takes care of a key is perfectly identified. This mapping defines a DHT, and is the basis of highly-scalable, distributed storage networks. In Fig. 1, we can see a possible mapping of a key space [0-10] in a 4-node DHT.

In a DHT, information is stored in the form of pairs {key, value} accessible by a hash-like API, providing us with functions to insert and modify key–value pairs (*put*(*key*, *value*)) and to access them (*value*=*get*(*key*)). Implicitly, a *routing protocol* will be used in order to deliver this petitions to the key owner node.

Different routing protocols have been proposed in the literature but all of them share one characteristic: the routing is done in a progressive manner, as a function of the distance to the destination node. Therefore, when a given node wants to communicate with the node that takes care of key k, it will send the message through the neighbor which is closer (in terms of assigned keys) to the destination. The specific definition of proximity of keys and nodes depends on the particular structure of the DHT, and varies from system to system.

In theory, DHT systems guarantee that any object can be reached in a number of jumps in order $O(\log N)$, being N the number of network nodes. The weakest point in DHT systems is their behavior in the case of fast changes in system configuration, term also known as *churn*. Informally, this refers to nodes that join or leave the overlay, forcing a network reorganization and a modification in the mappings. Latency can increase in this case, and several proposals exist that try to reduce this problem. In [13], an algorithm is proposed that tries to reach the optimum latency in potential-law graphs, such as P2P networks, without losing the scalability of DHTs; also, Godfrey et al. propose in [14] an algorithm to maintain load balance in adverse conditions.

3. P2P computing

A distributed computing system can be defined as a collection of computers interconnected by a communication network. These computers try to join their resources in order to collectively do computational tasks. Each computer in the system has its own, independent resources; however, from the user's point of view, the system should be seen as a single *resource pool*. An interface is given to the users in order to access the system without taking care of its complexity.

The are many systems, both free and commercial, that accomplish this objective, from *Clusters* to *Grid Computing* systems, and including *Desktop Computing* systems. Some well-known products or middleware sets are Globus [15] for grid systems, Condor [7] for cluster systems (with extensions for grids), or Boinc [8] for desktop computing. All of these systems, however, have something in common: they all revolve around central points of administration. This central resource, in case of failure, can make the whole system unusable.

A true peer-to-peer computing system overcomes this disadvantage by distributing management capabilities among all the system nodes. In the literature, we can find several proposals of P2P computing systems. An extensive survey can be found in [10]. Next we briefly analyze some P2P computing systems paying attention to the following desirable properties:

1. Fully distributed, without centralized administration point or points.
2. Users should be able to submit jobs from their machines and then disconnect from the system.
3. Global scheduling policies should be implemented. In particular, FCFS execution order of jobs is expected, although it needs not be strict.

Some proposals discussed in the literature fail to meet the first property: they use some centralized mechanism for job scheduling. CompuP2P [16] is based on a DHT which divides the nodes set between resource sellers and buyers. A *leader* node is chosen, which takes care of the resource market. In [17] Chmaj et al. present a system based on a structured overlay in which one of the nodes has the master role, the tracker. This tracker is needed in order to schedule computing tasks and distribute those files the nodes may require.

A common model used for P2P computing systems is the *super-peer* model in which a group of nodes, the super-peers, form a P2P overlay. Workers are connected to super-peers, which are in charge of scheduling tasks. When a user wants to run any task, he will have to ask nodes in the super-peer overlay to search for idle workers. Examples of this model are JACEP2P-V2 [18], Mining@home [19] and DIETj [20]. A similar idea is implemented in CoDiP2P [21], which is based on a tree-structured P2P network. It is built with the JXTA Java library, used to form non-structured P2P networks. In the tree structure, nodes are separated into masters and slaves, and grouped in tree regions. In each region, there is a master