Towards a Data-driven Coordination Infrastructure for Peer-to-Peer Systems

Nadia Busi, Cristian Manfredini, Alberto Montresor, and Gianluigi Zavattaro

[1] Dipartimento di Scienze dell'Informazione, Università di Bologna,
Mura A.Zamboni 7, I-40127 Bologna, Italy.
E-mail: {busi,manfredi,montresor,zavattar}@cs.unibo.it

Abstract. Shared dataspaces, initiated by Linda since the beginning of the 80s, has been successfully adopted as a coordination model in a huge variety of systems and applications, going from parallel computing to web-based collaborative work. We point out several scalability problems which arise when trying to exploit the original Linda coordination model in peer-to-peer systems. The objective of this analysis is to produce some guidelines for the design of a data-driven coordination infrastructure suitable for the peer-to-peer scenario.

1 Introduction

The rapid evolution of computers and networks is calling for the development of middleware platforms responsible for the management of dynamically reconfigurable federations of devices, where processes cooperate and compete for the use of shared resources. In this scenario one of the most challenging topics is concerned with the coordination of the activities performed by the federated components.

Generative communication, realized by means of the insertion and withdrawal of elements from a shared multiset, is the peculiar feature of a family of coordination languages, of which Linda [Gel85] is the most prominent representative. Generative communication is based on the following principles: a sender communicates with a receiver through a shared data space (called *tuple space*, TS for short), where emitted messages are collected; the receiver can consume the message from TS; a message generated by a process has an independent existence in the tuple space until it is explicitly withdrawn by a receiver; in fact, after its insertion in TS, a message becomes equally accessible to all processes, but it is bound to none.

In the last decades, the shared dataspace approach has been successfully adopted in a huge variety of systems and applications, going from parallel computing to Web–based collaboration system. Recently, this communication mechanism has been adopted also by several proposals of coordination platforms (see, e.g., Sun Microsystems JavaSpaces [W^+98] or the IBM T Spaces [WMLF98]) for the management of dynamically reconfigurable federations of devices, where processes cooperate and compete for the use of shared resources.

In this paper we investigate the scalability of this coordination approach to the realm of peer-to-peer systems.

Informally, *peer-to-peer* (P2P) systems are distributed systems based on the concept of resource sharing by direct exchange between *peer* nodes (i.e., nodes having the same role and responsibility). Exchanged resources include content, as in popular P2P file sharing applications [Shi01,Kan01,Lan01], and storage capacity or CPU cycles, as, for example, in computational and storage grid systems [And01,RD01,K$^+$00].

Distributed computing was intended to be synonymous with P2P computing long before the term was invented, but this initial desire was subverted by the advent of client-server computing popularized by the World Wide Web. The modern use of the term P2P and distributed computing as intended by its pioneers, however, differ in several important aspects. First, P2P applications reach out to harness the outer edges of the Internet and consequently involve scales that were previously unimaginable. Second, P2P by definition, excludes any form of centralized structure, requiring control to be completely decentralized. Finally, and most importantly, the environments in which P2P applications are deployed exhibit extreme dynamism in structure, content and load. The topology of the system typically changes rapidly due to nodes voluntarily coming and going or due to involuntary events such as crashes and partitions. The load in the system may also shift rapidly from one region to another, for example, as certain files become "hot" in a file sharing system; or the computing needs of a node suddenly increase in a grid computing system.

2 Shared Dataspaces in Mobile Systems

The pervasiveness of the client-server architecture also affected the design of shared dataspace based coordination infrastructures. Indeed, in most of the currently available implementations of Linda-like systems, the dataspace metaphor is intended as a (centralized) repository service.

An interesting proposal breaking the client-server bias is represented by the *transiently shared dataspace* metaphor introduced in Lime [PMR99].

Lime [PMR99] (Linda in a Mobile Environment) is a coordination middleware supporting both logical and physical mobility. It provides programmers with a Linda-like dataspace, whose content is determined by the connectivity among mobile hosts.

It is reasonable to investigate the scalability of Lime to a peer-to-peer context, characterized by dynamically changing connectivity, significant autonomy for the processes and direct communication between them.

The following discussion takes into consideration three different aspects of coordination models: *coordinables* (what is coordinated), *coordination rules*, *coordination medium*.

As far as coordinables are concerned, in a Linda-like system, the coordinated entities are usually active programs called agents. In Lime, agents with their own local dataspace reside on hosts and are able to logically move from a host to another one; the hosts themselves can physically move. In a peer-to-peer scenario, the involved entities (peers) are dynamic, i.e., they can frequently connect and disconnect, but they are not necessarily mobile. Mobility is not a peer-to-peer requirement in general.

With regard to the coordination rules, we can distinguish between two different programming styles: *context aware programming* and *context transparent programming*. Context aware applications are those which access both the system configuration context and the data context explicitly. For example a piece of new data may be stored on a specific mobile host. In contrast, context transparent applications can be developed without explicit knowledge of the current context. Peer-to-peer applications do not need to know where a resource is located, they only need to know if the resource is available. From this point of view, it is advisable to avoid context aware programming and replicate resources in order to improve their availability.

In Lime, the coordination medium is accomplished via a *transiently shared dataspace*. The dataspace content is determined by the connectivity among the mobile hosts. This kind of coordination medium supports physical and logical mobility of host and agents, respectively, and guarantees the consistency of the global data structure. As already discussed in [BZ01,BCV01], the consistency assumptions taken in Lime require an agreement among the involved entities regarding the set of federated hosts. For this reason, the system does not scale satisfactorily to a peer-to-peer scenario in which a clear partition of the peers in distinct clusters is unavailable and, moreover, the connection topology may be highly dynamically reconfigurable.

3 Shared Dataspaces in Peer-to-peer Systems

In this section we present some guidelines permitting to solve the scalability problems envisaged in the previous section. To be as general as possible, in this analysis we abstract away from the internal structure of the shared data. In this way, the proposed model can be instantiated to deal with any form of data, such as, e.g., Linda-like tuples, XML documents or JavaSpaces-like entries. Regarding the structure of the global dataspace, we assume that data are not correlated one to each other, i.e., the dataspace can be considered as a bag of independent items. This features differentiate our vision from other approaches, such as PeerWare [CP02], in which an overall tree-like structure is imposed on the collection of data.

The guidelines proposed in this work can be classified into two groups: those concerned with the production of data and those related to data retrieval.

As far as the nature of shared data in a P2P context is concerned, we single out two new classes of data that a peer can produce, besides the typical Lime-like, location aware form of datum.

As in Lime, also in P2P systems it may be useful to locate a new datum on a specified peer. This feature is useful to model resources or information which are strictly connected to an entity of the system, hence they must disappear when the entity becomes disconnected. As an example, consider data which represent resources or services provided by a peer.

As discussed in the previous section, we think that in this setting it is useful to provide abstractions for context transparent data sharing. We characterize at least two forms of such data.

The first step towards context independence is represented by *generic data*. In the spirit of the generative communication approach, a datum belonging to this class has an existence which is completely independent from its producer. Hence, the coordination infrastructure may decide to locate this datum in any of the available storages, as well as to move the datum according to some system or application specific needs. As an example, consider load balancing or accessibility improvement. An interesting aspect related to this form of datum is the so called time- and space-uncoupling: data are accessed independently of both the time when they are produced and the peer which created them. This kind of datum can be useful, e.g., to achieve a form of disconnected master-worker interaction, in which the involved entities connect to the P2P system only when they need to produce or consume job requests, which are processed while disconnected.

A further step towards context transparency can be performed in the case the datum represents an information or a resource that cannot be consumed. Clearly, these data can never be explicitly removed from the repository. Because of this feature, the coordination infrastructure can transparently replicate the datum in order to improve its availability to the peer community, as well as fault tolerance. We refer to this class as *replicable data*. A typical application which surely benefits from this kind of data is represented by file sharing systems.

Concerning data retrieval mechanisms, we observe that it is useful to provide the peers the possibility to define their own visibility horizon of the system, instead of forcing a predefined scope, as it happens, e.g., in Linda and in Lime. In fact, in the first case the scope coincides with the whole dataspace, while in the second one the scope is formed by the union of the contents of the repositories of the currently federated hosts. This idea may be realized by equipping each retrieval operation with an extra parameter, specifying the actual scope to be used. A reasonable metric for scope definition is the Time To Live (TTL), corresponding to the relative distance between the peer hosting the datum and the peer performing the operation in the current topology. This feature adds both inter- and intra-peer flexibility. More precisely: two different peers may have different visions of the global dataspace and the same peer may change its own scope in different retrieval operations.

4 Conclusion and Future Work

In this paper we initiate the design of a data-driven coordination infrastructure suitable for P2P systems; more precisely, we discuss some useful guidelines we intend to follow in the design of this infrastructure.

We plan to continue this line of work along two tightly related directions. On the one hand, we plan to develop a formal specification of the infrastructure, useful to clarify possible design ambiguities as well as to provide a formal framework for property verification.

On the other hand, we intend to verify the feasibility of the proposed model by implementing a prototype based on JXTA [jxt], an open-source P2P project promoted by Sun Microsystems. JXTA is aimed at establishing network programming platform for P2P systems by identifying a small set of basic facilities necessary to support P2P applications and providing them as a building block for high-level functions.

The choice of basing our implementation on JXTA has several benefits. First of all, the JXTA core provides the possibility of using different transport layers for communication, including TCP/IP and HTTP, and is capable of handling firewall- and NAT-related problems. Furthermore, the peer discovery mechanism included in JXTA will be used to enable peers to discover each other in the network and merge their dataspaces. Finally, we will exploit the complex security architecture that is being developed for JXTA, in order to add security mechanisms to our implementation.

References

[And01] D. Anderson. SETI@home. chapter 5. March 2001.

[BCV01] M. T. Valente B. Carbunar and J. Vitek. Lime Revisited. Reverse Engineering an Agent Communication Model. In *Proc. of MA'01*, Lectures Notes in Computer Science. Springer-Verlag, Berlin, 2001.

[BZ01] N. Busi and G. Zavattaro. Some Thoughts on Transiently Shared Dataspaces. In *Proc. on the Workshop on Software Engineering and Mobility (at ICSE 2001)*, 2001.

[CP02] G. Cugola and G.P. Picco. Peerware: Core middleware support for peer-to-peer and mobile systems. Draft - 2002.

[Gel85] D. Gelernter. Generative Communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985.

[jxt] Project JXTA. http://www.jxta.org.

[K+00] J. Kubiatowicz et al. OceanStore: An Architecture for Global-Scale Persistent Storage. In *9th International Conference on Architectural support for Programming Languages and Operating Systems*, Cambridge, MA, November 2000.

[Kan01] G. Kan. Gnutella. chapter 8. March 2001.

[Lan01] A. Langley. Freenet. chapter 8. March 2001.

[PMR99] G.P. Picco, A. Murphy, and GC. Roman. Lime: Linda Meets Mobility. In *Proc. 21th IEEE Int. Conf. on Software Engineering (ICSE)*, pages 368–377, 1999.

[RD01] A. Rowstron and P. Druschel. Storage Management and Caching in PAST, a Large-Scale, Persistent Peer-to-Peer Storage Utility. In *18th*, Canada, November 2001.

[Shi01] C. Shirky. Listening to Napster. chapter 2. March 2001.

[W+98] J. Waldo et al. Javaspace specification - 1.0. Technical report, Sun Microsystems, March 1998.

[WMLF98] P. Wyckoff, S. McLaughry, T. Lehman, and D. Ford. T spaces. *IBM Systems Journal*, 37(3):454–474, 1998.