J. Parallel Distrib. Comput. 78 (2015) 6–24

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

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

A load balanced directory for distributed shared memory objects☆

Gokarna Sharma *, Costas Busch

School of Electrical Engineering and Computer Science, Louisiana State University, Baton Rouge, LA 70803, USA

HIGHLIGHTS

- A novel load balanced directory for distributed shared memory objects is proposed.
- It is suitable for d-dimensional mesh-based topologies with n nodes and $d \geq 2$.
- It attains $\mathcal{O}(d^2 \log n)$ load approximation and $\mathcal{O}(d \log n)$ stretch approximation.
- Experimental results confirm the load balancing and low stretch benefits.
- Previous protocols only considered stretch and cannot control network load.

ARTICLE INFO

Article history:
Received 29 May 2014
Received in revised form
14 October 2014
Accepted 12 February 2015
Available online 23 February 2015

Keywords:
Distributed systems
Distributed directory
Shared object
Cache-coherence
Mesh network
Load balancing
Stretch
Oblivious routing

ABSTRACT

We present MultiBend, a novel distributed directory protocol for shared objects, suitable for large-scale distributed shared memory systems that use d-dimensional mesh-based topologies, where $d \geq 2$. Each shared object has an owner node that can modify its value. The ownership may change by moving the object from one node to another in response to *move* requests. The value of an object can be read by other nodes with *lookup* requests. MultiBend balances the load of the network edges and nodes by forwarding each *move* or *lookup* request and response along a path consisting of multiple bends in the mesh. Using an oblivious routing protocol, the multi-bend paths have a small number of overlaps which helps to reduce the maximum edge and node utilization to achieve load balancing. At the same time, MultiBend achieves small stretch for the total path length of any sequence of *move* requests, compared to the total optimal path length. MultiBend guarantees $\mathcal{O}(d^2 \log n)$ approximation for the load, and $\mathcal{O}(d \log n)$ approximation for the stretch due to *move* requests, where n is the number of nodes in the mesh network. It also guarantees $\mathcal{O}(d^2)$ approximation for the stretch of *lookup* requests. We evaluate MultiBend with simulations using various sequences of *move* and *lookup* operations in a 16×16 nodes 2-dimensional mesh network. We compare the simulation results to other protocols which are not tailored for load balancing and we find that our protocol is better by as much as the factor of 6.85 in terms of congestion in the worst-case. To the best of our knowledge, this is the first distributed shared memory directory protocol that considers the network load balancing aspect and achieves good approximation ratio for both the load and the stretch.

1. Introduction

Many distributed systems rely on some concept of *objects* which are the individual entries at the shared memory available in these distributed systems. An object can be shared by multiple tasks on different network nodes. A *task* is a sequence of shared memory operations (i.e., reads or writes), and the nodes are the processors in distributed systems which communicate through a message passing environment. We assume that each processor has its own *cache*, where copies of objects reside. When a task running at a processor node issues a read or write operation for a shared memory location, the data object at that location is loaded into the processor-local cache.

We consider distributed systems where shared objects are moved to those nodes that need them. In other words, tasks at network nodes can only operate on local shared objects and, if remote shared objects are required, the task must communicate with one or more remote processor nodes. Some distributed *cache-coherence* mechanism should ensure that shared objects remain *consistent*, i.e., writing to an object automatically *locates* and *invalidates* other cached copies of that object. These mechanisms are widely known as *distributed directory protocols* (DDPs) in the literature, e.g., [9].

☆ This paper extends a preliminary version that appears in the *Proceedings of the 18th International European Conference on Parallel and Distributed Computing* (Euro-Par 2012) Sharma and Busch (2012) [24].

* Corresponding author.
E-mail addresses: gokarna@csc.lsu.edu (G. Sharma), busch@csc.lsu.edu (C. Busch).

http://dx.doi.org/10.1016/j.jpdc.2015.02.002
0743-7315/

A DDP typically supports three kinds of operations: (i) *publish* operation, which is used by a node that creates an object to enable other nodes to find that object; (ii) *lookup* operation, which is used by a node to search for an object and retrieve a copy if the object exists; and (iii) *move* operation, which is used by a node to retrieve an object from another node. Any DDP guarantees that each *lookup* and *move* to the shared object is individually *atomic*.

DDPs have a long history of research. They have widely been used in distributed shared memory implementations in multi-cache systems, e.g., [9,2,8,11]. Very recently, DDPs have been studied for transactional memory implementations in large-scale distributed systems, e.g., [5,14,24,26,28], and are called *distributed transactional memory consistency protocols* (DTMs). DTMs act similarly as of DDPs when there is one shared object in the system or when all the transactions[1] in the system are operating on only one shared object (not necessarily the same shared object). However, in the case when transactions are operating on many shared objects, they may *conflict* with each other in accessing the shared objects. The conflicts between any two transactions are resolved either by aborting a transaction or by postponing operations of one transaction for a fixed duration, so that the other transaction gets chance to commit. In other words, in addition to the capabilities of DDPs, DTMs should have the mechanism to support aborts of transactions. This is generally provided with the help of a globally consistent contention management policy which determines which transaction to abort among the conflicting transactions. A number of policies are proposed in the literature, e.g. [12,4,22]. Note that if more than two transactions are conflicting in accessing multiple objects then the contention management policy should guarantee that there does not occur any deadlock and livelock of transactions. DTMs that are proposed in the literature [5,14,26,28] are basically DDPs as they do not provide full support for transaction aborts and most of them consider only one shared object.

Typically the performance of a DDP is measured with respect to the communication cost, which is the total number of messages sent in the network. The total number of messages sent by an operation is measured with respect to the number of edges of the network used by that operation. The communication cost for an operation (respectively for a set of operations) is compared to the optimal communication cost for that operation (respectively for that set of operations) to provide an approximation ratio (or a competitive ratio) on communication cost, which is generally referred to as *stretch*. In the context of DDPs, previous approaches: Arrow [11], Relay [28], Combine [5], Ballistic [14], and Spiral [26], focused only on stretch bounds for various network topologies and they do not control the congestion. Moreover, DDPs designed and used in [9,2,8] have not been analyzed even for the stretch. The network congestion can also affect the overall performance of the algorithm and sometimes it is a major bottleneck. We measure the network congestion as the worst node or edge utilization (the maximum number of times the object requests use any edge or node in the network while accessing the shared object).

1.1. Problem statement

Given a network and a set $\mathcal{E} = \{r_0, r_1, \ldots, r_l\}$ of $l + 1$ object operations (l does not need to be known and the bounds are independent of l), where r_0 is the initial *publish* operation and the rest are the subsequent *move* operations. DDPs organize object operations into a total order (or a "distributed queue") [11]. Note that we are focusing here on *move* operations only; the reason behind not having *lookup* operations in $\mathcal{E}$ is that *lookup* operation do not change the way DDPs form distributed queues, i.e., they do not modify the queue formed by *move* operations. Each operation r_i has a source node s_i and a destination node t_i. (Source nodes may be different than object creating nodes; object creating nodes and the nodes that currently have the objects are denoted by *owner nodes*.) In the DDP problem, the destination node of an operation r_{i_1} is the source node of another operation r_{i_2} in the total order, where the total order is a permutation of the requests in $\mathcal{E}$. That is, the destination node for each operation is not known beforehand and the DDP should find out the destination node online while in execution. The goal is to find a path p_i from s_i to t_i, for every request r_i, while minimizing both the maximum congestion along any edge e (any node v) in the network and the communication cost (the number of edges e that p_i uses). Formally,

- *Load balancing:* Minimize total edge congestion $C = \max_e |\{i : e \in p_i\}|$ and total node congestion $C_n = \max_v |\{i : v \in p_i\}|$. The congestion C (C_n) can be compared to the optimal congestion C^* (C_n^*) that is attainable by any DDP to provide an approximation ratio on congestion for any edge e (any node v). Congestion on network edges and nodes can adversely affect the overall performance of the algorithm, especially in systems with limited bandwidth and/or in systems with limited computation power. For example, in sensor networks congestion can lead to random dropping of data and dramatic increase in energy consumption [15]. Congestion minimization is very important because it allows to evenly utilize available network resources (edges and nodes), avoiding the chance of the system being bottleneck due to some "hotspot" resources. This is done by reducing the communication/computation load on network edges and nodes through load distribution optimization.
- *Stretch:* Minimize total communication cost $A(\mathcal{E}) = \sum_{i=1}^{l} |p_i|$, where $|p_i|$ is the number of edges that the path p_i of the request r_i uses. The total communication cost $A(\mathcal{E})$ can be compared to the cost $A^*(\mathcal{E})$ of an optimal offline ordering algorithm OPT that has complete knowledge about all the requests in $\mathcal{E}$ to provide the stretch, i.e., the stretch is the ratio $A(\mathcal{E})/A^*(\mathcal{E})$. We are interested to minimize $\max_{\mathcal{E}} A(\mathcal{E})/A^*(\mathcal{E})$, the maximum ratio over all sets of operations $\mathcal{E}$.

1.2. Contributions

We present MultiBend, a DDP for shared objects, that is suitable for d-dimensional mesh networks, where $d \geq 2$, and is load balanced in the sense that it has low congestion (maximum edge utilization), and at the same time maintains low stretch. Mesh networks are appealing due to their use in parallel and distributed computing [17,1,10,23]. Mesh networks are cost-effective and provide great performance solutions for diverse applications, simple expansion for future growth, and scalable connection properties. Mesh topologies are used as an underlying backbone network in many distributed clusters and supercomputers. For example, 65,000 nodes of IBM Blue Gene/L are interconnected as a $64 \times 32 \times 32$ 3-dimensional mesh or torus [1]. Recently, IBM Blue Gene/Q integrated 5-dimensional torus [10], where a torus is a variation of the mesh.

MultiBend combines in a novel way a DDP protocol with a routing algorithm to achieve low stretch and load balancing. The low stretch is achieved through a hierarchical directory which we first introduced in [26] for general networks and we adapted here for the mesh network. The load balancing is achieved through an *oblivious routing* approach (e.g., [17,6,7]) tailored to the d-dimensional mesh; in particular, we use the oblivious routing algorithm in Busch et al. [7]. A routing algorithm is *oblivious* if every path that is selected for each request to route to its destination

[1] A transaction represents a sequence of shared memory operations (i.e., reads and writes) that are all performed atomically.