Distributed Flow Control and Intelligent Data Transfer in High Performance Computing Networks

Hochschule Offenburg

Mehdi Sadeghi

13. Feb 2015

Declaration

I hereby declare and confirm that this thesis is entirely the result of my own original work. Where other sources of information have been used, they have been indicated as such and properly acknowledged. I further declare that this or similar work has not been submitted for credit elsewhere.

Offenburg, February 28, 2015
Mehdi Sadeghi

Abstract

This document contains my master's thesis report, including the problem definiation, an overview of state of the art, discussions and my suggestions.

Acknowledgement

Here will come acknowledgement

Contents

Preface

European scientific communities launch many scientific experiments daily, resulting in huge amounts of data. Specifically in molecular dynamics and material science fields there are many different simulation software which are being used to accomplish multiscale modelling tasks. These tasks often involve running multiple simulation programs over the existing data or the data which is produced by other simulation software. Depending on the amount of the data and the desired type of simulation these tasks could take many days to finish. The order to run simulation software and providing the input data are normally defined in scripts written by researchers which is the simplest form of workflow management.

While small experiments could be handled with simple scripts and normal computers, larger scale experiments require different solutions. These type of experiments demand huge computing power which are made available by computer clusters and super computers. An important chracteristic of larger experiments is the amount of produced data. This data, which needs to be transferred many times back and forth between computers, grow by an order of magnitute, resulting in terabytes of data.

Large scientific experiments are the source of many high performance computing (HPC) problems, specially data transfer and workflow management. Moreover HPC resources are expensive and should be used efficiently, therefore making data transfer more efficient is important.

This thesis is an effort to know the main data transfer scenarios in HPC experiments and try to address them in a distributed manner with a collective but decentralized approach toward workflow management.

If there are any comments and improvements regarding this document, the author appreciates an email to the following address:

`msadeghi@stud.hs-offenburg.de`
Hochschule Offenburg
Mehdi Sadeghi

Chapter 1

Introduction

1.1 Objectives

We will focus on two important topics in this work. First one is data transfer problems between multiple computers which are doing a task collaboratively. Second one is the collaboration itself, i.e. how multiple computers will accomplish a collaborative task in a distributed and decentralized environment. To accomplish these objectives we will discuss our specific distributed work flow management and data transfer scenarios. We define our requirements regarding the above mentioned topics and we will extract the parameters which we are going to assess other solutions with them. Then we will go through the currently available solutions. Afterward we will discuss their applicability according to our requirements and whether they answer our needs or not. The goal of this thesis is to minimize the amount of transferred data in a network of collaborating computers. This data belongs to operations which might be linear or not and require multiple steps. The operation can be initiated in any participating computer but the required data is not necessarily available on that computer, even though the operation result should be delivered back to the reinitializing computer.

1.2 Terminology

We will use a number of terms through this report. Here are the meaning for each.

Node Each node refers to one computer in the network.

Data When we refer to data we mean the output of scientific applications, such as NumPy types.

Dataset Same as data with more emphasize on it as collection of e.g. NumPy types.

Application Refers to the demo application which has been developed to show case the proposed solution.

Instance Refers to an instance of the same application running on a node.

Operation Any sort of operation, linear or non-linear which is being provided by the application.

Task Same as the operation with more emphasize on the output rather than the functionality.

Service A scientific operation being provided by the application which could be called remotely.

System The combination of nodes, data, application, instances, operations and services as a whole.

User A scientist, researcher or student who uses the system to run a task.

1.3 Problem Context

Whole this work is an effort to address issues of a scientific environment. Some particular characteristics are running multiple scientific programs on different computers which need to exchange data in order to accomplish one operation. Another task which is often done is visualization. Visualizing the operation results ,depending on the requested visualization, might require heavy computational tasks i.e. average or comparison on data which might not be available on the same machine or might be residing partially on different computers. The produced data often exceeds 1 GB in many experiments and it should be moved back and forth every few minutes, therefore it is cheaper to transfer the operation rather than the data.

The problem here is not about distributing the stored data rather, data exchange between instances of the application talking together in runtime while doing one global task and keeping this workflow distributed. In this terms each application instance takes care of its own data and provides a set of services. Some operations require data from another node, therefore we have to transfer the data or run the operation on the node which contains the data. There are a number of scenarios which we will discuss.

1.4 Assumptions

During this work we have a number of assumptions. We have a certain problem which we want to focus on rather than reintroducing solutions that already exist. For this reason we discuss regarding our needs.

1.4.1 Collaborating Network

We assume there is a network of computers which are available to run the tasks, each node is running an instance of the application. We will propose our collaboration and data transfer algorithm between them later.

1.4.2 Data Characteristics

We need to discuss more about the data. In our scientific context data is mostly numerical and explains characteristics of physical particles such as atoms and molecules. These data is being used to simulate collections of particles called models. Although our work is not dependent on these, they help us to understand the the definition of the data that we often refer to in this report. One important aspect of the data that we are interested in is that it is not critical and we can reproduce it.

1.4.3 Workflow

In contrast to data we are interested in workflow. We want to find a reliable approach to access and update state of our workflow on any arbitrary node which is part of our collaborative network.

1.4.4 Data Transfer

We assume a data transfer approach is already in place. This could be any file system which supports network storage. Rather than going into details

of how data could be transferred more efficiently, we will focus on finding which data to be transferred and from which computer to which destination.

1.5 Scenarios

There are a number of possible use cases according to the desired operations. To demonstrate these cases we assume we have four nodes, A, B, C and D and four datasets respectively, but not necessarily on the respective nodes. In the following paragraphs we explain possible combinations of operations, nodes and datasets. In every scenario we want to run an operation which could be linear or non-linear and we need data for that operation which could be on the same node that runs the operation initially or could reside on many other nodes. Moreover there is output dataset which should be stored.

We start from the simplest scenarios first, it means one linear atomic operation which needs only one dataset to operate on it.

1.5.1 Decision Making

The main decision that we need to make at every scenario is whether we should transfer the required data or we need to delegate the operation to an instance on a node which already has the data. To make a decision we need to answer a number of questions. First we need to know the location of the data and whether we can produce it or not:

1. Is the data available locally?
2. If not, is the data available on another node? – Here only the physical location of data matters not the instance controlling it.
3. If not, can we generate the data? – should we take data generation into account at all?
4. If not, which instance can generate the data?

In case the mentioned data is available on another nodes we have to answer these questions:

1. What is the cost of data transfer? – We have to invent an algorithm for this calculation
2. If data is available on more than one remote node, which one has the minimum transfer cost? – We might introduce multiple strategies and

use some heuristics for this selection, simplest form would be random selection, another could be asking for an availability metric from the instance and mix it with local calculated availability metric to get a final cost value.

3. Does the requested operation available on the other node?

4. In case it is expensive to transfer the date, can we delegate the operation to an instance on the other node?

Metrics

1.5.2 Scenario 1

In this scenario we have received a request for a linear atomic operation, e.g. Op^A on $Node^A$ which requires $Dataset^1$.

Conditions

$Dataset^1$ is not available on $Node^A$.

Consequences

With these conditions we either should transfer $Dataset^1$ to local node or in case of availability delegate Op^A to the node which already has $Dataset^1$.

Chapter 2

State of the Art

In this chapter we will go through a number of existing solutions and we will disscuss their efficiency and deployment complexity. Before that we introduce the parameters which are important for us. Then we will assess each solution against the introduced parameter set.

2.1 Parameters

There are many factors that we need to take into account before introducing our parameters. To name a few:

- Data transfer cost
- Data reproduction cost
- Type of operation on data, linear or non-linear
- Replication
- Deployment complexity
- Fault tolerance
- Portability
- Performance according to number of distributed nodes and size of files

2.2 Data Storage Systems

SAME POINTS SHOULD BE DISCUSSED FOR EACH APPROUACH (FROM OUR POINT OF VIEW) e.g. EFFICIENCY, COMPLEXITY, DISTRIBUTED APPLICATION ACCESS, APPLICATION AWARENESS, POSSIBLE DATA ACCESS SCENARIOS, SCALIBILITY, DATA TRANSFER, FAULT TOLERANCE, ACCESS CONTROL

2.3 Distributed File Systems

One way to achieve fault tolerant and reliable data storage and access is to use distributed file systems (DFS). In this case the data will be replicated over a network of storage servers with different magnitutes based on the underlying file system. We will discuss a number of free and open source solutions.

2.3.1 Hadoop Distributed File System (HDFS)

"The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware." [7, tp. 3]

"Hadoop1 provides a distributed filesystem and a framework for the analysis and transformation of very large data sets using the MapReduce [3] paradigm."[6]

"HDFS stores metadata on a dedicated server, called the NameNode. Application data are stored on other servers called DataNodes."[6]

Deployment Complexity src:http://hadoop.apache.org/docs/r0.18.3/quickstart.html needs Java 1.5.x ssh and sshd and rsync. Three basic modes are available: Local, Pseudo-Distributed and Fully Distributed mode. XML configuration, installation of Local and Pseudo Distributed modes are almost straight forward, for fully distributed note extra steps are required (offical doc link is dead).

Efficiency

Fault Tolerance "Hardware failure is the norm rather than the exception." "Each DataNode sends a Heartbeat message to the NameNode periodically." "The DataNodes in HDFS do not rely on data protection mechanisms such as RAID to make the data durable. Instead, like GFS, the file content is replicated on multiple DataNodes for reliability." [6]

Portability "HDFS has been designed to be easily portable from one platform to another."

Robustness "The primary objective of HDFS is to store data reliably even in the presence of failures."

Accessibility

1. FS Shell
2. DFSAdmin
3. Browser

Applicability There is a good document here: http://hadoop.apache.org/docs/r0.18.0/hdfs_design.pdf Hints: HADOOP is for big data and the programming should be different (map/reduce) and it does not look suitable for our use cases and requirements. The burden would be so high that we will waste a lot of resources. I have to put these in scientific words with more logic and references to sizes that we need and more numbers.

Users have to program their applications using Java and Hadoop to take advantage of distributed computing features in Hadoop MapReduce and HDFS. Cites? Hadoop website? https://infosys.uni-saarland.de/publications/BigDataTutorial.pdf

2.3.2 XTREEMFS

Chapter 3

Rough Ideas

This chapter contains very raw ideas to address main requirements i.e. distributed workflow management and intelligent data transfer.

3.1 Intelligent Data Transfer: Use Case One

By *Intelligent Data Transfer* we mean an approach that minimizes required data transfers between application[1] instances.

In the most basic use case[2] we run a script [3] which consists of two linear operations. Each operation consumes data and generates data. A third operation needs both generated data two operate on and generate the third and final data.

The script is data driven. It means that it contains a number of steps and for every step it needs appropriate data to run the desired operations[4]. We assume that the script will run on *Node 1* and required data *DataSet1* and *DataSet2* are located on *Node 2* and *Node 3* respectively. Therefor *Node 1* have to initiate operations on the other machines.

3.1.1 Identical Instances

We assume that on each machine of the network the same instance of our imaginary program is running which is capable of running all operations including A, B and C. The only consideration is the availability of DatSets, they are not available on all machines.

[1] To be defined
[2] To be added later and referenced here
[3] To be defined and added to the terminology, terminology itself has to be defined
[4] To be defined

A linear operation (not clear to my self how to write it):

$$Operation(A, B) = Operation(A) + Operation(B)$$

$$DataSet^A = Operation^A(DataSet^1)$$

$$DataSet^B = Operation^B(DataSet^2)$$

$$DataSet^C = Operation^C(DataSet^A, DataSet^B)$$

Assuming that operations A and B will run on the machines which contain the required data, a number of questions arise here:

1. On which machine operations C should run? A, B or C?
2. On How to transfer the required data to that machine in an optimized way?

3.1.2 The Idea

First of all we assume that we have the information about the DataSets available on all of the mamchines i.e. in form of a distributed table with entries containing the node address and DataSet id. Based on this information the application can decide if it has the required data or not.

Based on this algorithm (to be defined) the initial application delegates operations to the other nodes (instances of the same program), where the data is available. Our distributed workflow manager (to be defined) will synchronize the information on these running operation and will lable the output data and will add it to the distributed data table.

After finishing operations A and B we will run operaion C in either of these nodes, because the required data is partially available on these nodes. Then we have to transfer the rest of the data to one of these nodes to run the operation C which needs both parts simultaniously.

Using Prior Art At this point we can take advantage of existing Distributed File Systems (DFS) to make the data available for operation C. We can then eliminate the complexity of data transfer between these two nodes and delegate it to existing distributed file systems. The main point is we don't rely on DFS for all of our decistion making part but we explicitely make the decition to run operation A and B on specific nodes and then for the last part we use a meta disk or universal disk concept to deliver the remaining data for operation C.

Chapter 4

Experiments

In this chapter we will go through a set of experiments to showcase the result of taking different approaches toward file transfer techniques. Meanwhile we will try a number of data distribution methods to see how they fit into our scenarios.

4.1 Distributed Hash Tables

Distributed Hash Tables (DHT) known best for their application to build torrent tracking software, let us to have a key/value store and distributed it in a decentralized way amount a network of peers.

MORE INFO HERE ON DHT, KAMEDLIA PAPER and IMPLEMENTATIONS

In our case to keep track of the available data on the network of collaborating peers, we can use a DHT table. Everytime an instance wants to find a dataset it should query the network of peers using one of the existing wrappers and implementations.

4.2 Test Results

Our tests show that even though DHT is fault-tolerant and reliable for file distribution, it is not adequete for our realtime requirement to find our required data. In one test we ran two peers, one on an Internet host and another one on local host. Here are the client and server codes:

```
1 from twisted.application import service, internet
2 from twisted.python.log import ILogObserver
3
4 import sys, os
```

```
5  sys.path.append(os.path.dirname(__file__))
6  from kademlia.network import Server
7  from kademlia import log
8
9  application = service.Application("kademlia")
10 application.setComponent(ILogObserver,
11     log.FileLogObserver(sys.stdout, log.INFO).emit)
12
13 if os.path.isfile('cache.pickle'):
14     kserver = Server.loadState('cache.pickle')
15 else:
16     kserver = Server()
17     kserver.bootstrap([("178.62.215.131", 8468)])
18 kserver.saveStateRegularly('cache.pickle', 10)
19
20 server = internet.UDPServer(8468, kserver.protocol)
21 server.setServiceParent(application)
22
23
24 # Exposing Kademlia get/set API
25 from txzmq import ZmqEndpoint, ZmqFactory, ZmqREPConnection,
26  ZmqREQConnection
27
28 zf = ZmqFactory()
29 e = ZmqEndpoint("bind", "tcp://127.0.0.1:40001")
30
31 s = ZmqREPConnection(zf, e)
32
33 def getDone(result, msgId, s):
34     print "Key result:", result
35     s.reply(msgId, str(result))
36
37 def doGetSet(msgId, *args):
38     print("Inside doPrint")
39     print msgId, args
40
41     if args[0] == "set:":
42         kserver.set(args[1], args[2])
43         s.reply(msgId, 'OK')
44     elif args[0] == "get:":
45         print args[1]
46         kserver.get(args[1]).addCallback(getDone, msgId, s)
47     else:
```

```
48          s.reply(msgId, "Err")
49
50  s.gotMessage = doGetSet
```

In the above example we have used *twisted* networking library[8] and one python implementation[1] of *Kademlia* DHT algorithm[4]. This will start a p2p network and will try to bootstrap it with another peer on the give IP address. Thereafter it will open another endpoint to expose a simple *get/set* method for the rest of application for communicating with the network.

HERE DESCRIBE ABOUT DHT IMPLEMENTATION AND TWISTED NETWORKING LIBRARY. ***MOST IMPORTANT*** ABOUT REPLICATION OF DATA, IS THERE ANY REPLICATION? WHAT IF A NETWORK NODE FAILS? ***RCP OVER UDP AND WORK THROUGH FIREWALLS***

The next part is a few lines of code to communicate with this network:

```
1  #
2  # Request-reply client in Python
3  # Connects REQ socket to tcp://localhost:5559
4  # Sends "Hello" to server, expects "World" back
5  #
6  import zmq
7
8  # Prepare our context and sockets
9  context = zmq.Context()
10 socket = context.socket(zmq.REQ)
11 socket.connect("tcp://localhost:40001")
12
13 # Set request
14 socket.send(b"set:", zmq.SNDMORE)
15 socket.send(b"the key", zmq.SNDMORE)
16 socket.send(b"the value")
17 print socket.recv()
18
19 # Get request
20 socket.send(b"get:", zmq.SNDMORE)
21 socket.send(b"the key")
22 print socket.recv()
23
24 # Invalid get
25 socket.send(b"get:", zmq.SNDMORE)
26 socket.send(b"not existing")
27 print socket.recv()
```

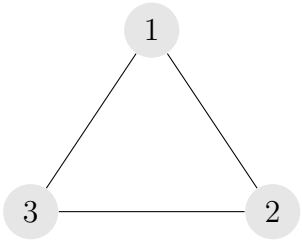

Figure 4.1: A network of three peers

This simple client will try to connect to the previously opened port and send get/set messages.

4.2.1 The Problem

Configuring this p2p network is a little tricky. The network should work correctly even if nodes enter and leave the network. During our tests in development environment we observed some problems with initializing the network, but while the network was initialized leaving and entering the network had no effect on the results.

Reliability

Having the number of nodes increased up to 3 the reliability shows up again. When we set a value for a key in one node we can not guarantee that getting the value for that key on other nodes will return the updated one. With a number of tests I can confirm that two nodes which are bootstrapped with the same third node does not provide the accurate result everytime and it is not clear for me why this happens. See figure 4.1 on page 19.

After running more tests, we figured out that the possible source of the above mentioned problems was the confusion in using *binary* and *string* in python, so it was an error in our side.

Firewall Problems

In a test having one process running on a server in Internet and outside of the local network and having two different processes running on one laptop but on different ports it is observed that the changes (sets) in the internet does not replicate to the local processes but the changes from local processes are being replicated to the other process.

4.2.2 conclusion

Having a network between local and internet processes in the above mentioned method is not reliable. Repeating the tests with only local processes which are bootstrapping to one of them and running the setter/getter methods showed that even in this scenario it is not reliable and one can not guarantee that the desired value will be returned.

References

[1] *A DHT in Python Twisted*. 2015. URL: https://github.com/bmuller/kademlia.

[2] Jared Bulosan Christopher Moretti et al. "All-Pairs: An Abstraction for Data-Intensive Cloud Computing". In: *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on* 11 (2008), pp. 352–358.

[3] Jeffrey Dean and Sanjay Ghemawat. *MapReduce: Simplified Data Processing on Large Clusters*. Proc. Sixth Symposium on Operating System Design and Implementation, 2004.

[4] Petar Maymounkov and David Mazières. "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric". English. In: *Peer-to-Peer Systems*. Ed. by Peter Druschel, Frans Kaashoek, and Antony Rowstron. Vol. 2429. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2002, pp. 53–65. ISBN: 978-3-540-44179-3. DOI: 10.1007/3-540-45748-8_5. URL: http://dx.doi.org/10.1007/3-540-45748-8_5.

[5] K. Ranganathan and I. Foster. "Decoupling computation and data scheduling in distributed data-intensive applications". In: *Proc. 2002 High Performance Distributed Computing IEEE International Symposium* 11 (2002), pp. 352–358.

[6] *The Hadoop Distributed File System*. URL: http://www.aosabook.org/en/hdfs.html.

[7] *The Hadoop Distributed File System: Architecture and Design*. 2007. URL: http://hadoop.apache.org/docs/r0.18.0/hdfs_design.pdf.

[8] *Twisted Matrix Project*. 2015. URL: https://twistedmatrix.com/.