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 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.

1.4.4 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.

Approaches

During this work we consider three different approaches toward preparing required data for operations.

Conventional Approach in this approach we put the required data on a network file system and all application instances will access it there. We will utilize an NFS mounted file system.

Centralized Approach in this approach we will have a central instance which will orchestrate operation delegation and operation output forwarding to other nodes.

Decentralized Approach in this approach we will eliminate the orchestrator node and the network of application instances should collaborate in a decentralized fashion to keep track of data and control flow for each task.

For every approach we will run performance tests and we will compare the results.

Method

We will discuss scenarios in 5. For each scenario we will analyze the possible combinations of data and operations and we will discuss how to deliver the input data and where to store output data. We will discuss workflow management in chapter 6 and data transfer in 7.

Chapter 2

Rough Ideas

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

2.1 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.

2.1.1 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?

2.1.2 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.

2.2 Sqmpy Integration

We can use Sqmpy project as a monitoring tool for konsensus network. Providing one peer address it can query the rest of peers and connect or subscribe to their news channel. Having this we can always see which nodes are offline and which ones are online. This also gives us a platform to extend monitoring and control features to the web. Currently we have made all the required software platform to achieve this. On Sqmpy side using Flask microframework and socket.io we can simply maintain realtime connections to the browsers and since our web framework is written in python, with minimum cost we can integrate it with konsensus.

Chapter 3

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.

3.1 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.

3.2 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[14] and one python implementation[1] of *Kademlia* DHT algorithm[5]. 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()
```

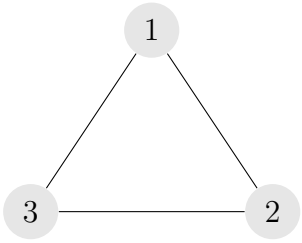

Figure 3.1: A network of three peers

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

3.2.1 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 3.1 on page 16.

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.

3.2.2 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.

3.3 Network Programming

To showcase our desired approach and trying different ones we have used a number of network programming frameworks for python programming language. The main library that we use is called ØMQ or ZeroMQ [**ZeroMQ**]. ZeroMQ is an asynchronous messaging library written in C with bindings for many languages including python. This library helps us to easily scale and use different programming paradigms such as publish-subscribe, request-replay and push-pull.

3.4 Publish-Subsribe Method

Because of reliability issues we fallback to using a simpler approach using ZeroMQ. In this stage our aim is to distribute the information about available datasets at each node. To achieve this we let our demo application launch a number of communicators and publish information about it's data. Other nodes in our network have to subscribes on other nodes, hopefully ZeroMQ allows us to subscribe to multiple publishers, therefore each node can subscribe to other nodes. Nodes frequently get **news** from other nodes, for example availability of certain datasets on a node, then it can use publish-subscribe to get extra information on that particular subject.

3.4.1 Architecture

For demonstration purposes we create a python console applicaiton using gevent[1], zeromq[2] and zerorpc[3] to be able to service multiple requests in a non-blocking fashion.

[1] http://www.gevent.org/
[2] http://zeromq.org/
[3] http://zerorpc.dotcloud.com/

Applicaiton Initialization

First of all each application instance establish its own zeromq publisher socket. Then it subscribes itself to all other nodes which are listed in config file. At this stage it should be configured manually.

Gevent and non-blocking

HERE WRITE ABOUT GEVENT.

3.4.2 Exposing API

Since this is going to be a network program we need to use a form of Remote Procedure Call (RPC) to communicate between nodes. Rather than implementing ourselves we used a library based on zeromq called *zerorpc*. Using this library we now expose a set of APIs and let the nodes talk to each other based on this API. There are multiple solutions for exposing services which we do not discuss here.

Data transfer using zerorpc

WHICH DATA TYPES ARE WE ALLOWED TO TRANSFER USING ZERORPCS? IS IT ENOUGH FOR US?

3.4.3 Application's State

WE HAVE TO DECIDE ON A STATE MANAGER TO MAKE APPLICATION'S BEHAVIOUR RELIABLE.

3.4.4 Network Discovery

At this stage there is no network discovery, because it is not our main problem. It can be done later as an improvement.

3.4.5 Failure Recovery

Again this is not of our interest. The point is there are existing solutions for these problems and we want to let our application to be able to demonstrate the main problem which would be deciding about data transfer routes and distributing the information about currently running operations.

3.5 Use Case One Problems

When we ask for an operation and we want to store the result somewehere on the network we have to think about the result name. We need a consistent way of naming datasets. If we ask users to provide resulting dataset names it will break soon, we need to let user to somehow give some **tags** but not the real names. We have to let the user know about the result name but also let her to look for datasets by providing some tags.

The simplest problems that will hapen if we store datasets with similar names is redandant work in the network. Peers will start to process and override the same dataset.

3.5.1 Data Manipulation

We will neet to let users to manipulate currently existing datasets, but very fast it comes to mind that not every dataset should be writable, we will need to categorize and identify our datasets based on some criteria. These problems are not part of my thesis but We mention it as part of problem analysis.

Chapter 4

Literature

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.

4.1 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

4.2 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

4.3 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.

4.3.1 Hadoop Distributed File System (HDFS)

"The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware." [13, tp. 3]

"Hadoop1 provides a distributed filesystem and a framework for the analysis and transformation of very large data sets using the MapReduce [4] paradigm."[12]

"HDFS stores metadata on a dedicated server, called the NameNode. Application data are stored on other servers called DataNodes."[12]

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." [12]

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

4.3.2 XTREEMFS

4.4 Distributed Objects

In this section we go through a number of existing methods to distributed an object or in another terms to distribute the state.

4.4.1 Concoord

Describe why it is not suitable for us. It allows single object sharing.

4.4.2 Distributed Hash Tables (DHT)

DHTs are known to be a distributed key/value storage.

Kademlia

Kademlia is a p2p DHT algorithm introduced in 2002.

4.5 Distributed Workflows

In this section we introduce a number of existing scientific workflow systems.

Chapter 5

Scenarios

There are a number of possible use cases in our problem domain. To demonstrate these cases we assume we have a number of nodes and datasets respectively, but they are not necessarily on the same nodes. In the following paragraphs we explain possible combinations of operations, nodes and datasets.

5.1 Operation

In every scenario we want to run an operation which could be linear or non-linear.

5.1.1 Linear Operation

Being linear means that the operation could be broken into its components and then run in parallel or series. Here is algebic notation of a linear operation which acts on two datasets:

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

Being linear or non-linear only matters when we have to operate on more that one dataset.

5.1.2 Non-linear Operation

In contrast to linear there is non-linear operation. This means that this kind of operation has dependant parts and those parts could not run in parallel:

$$Operation(A + B) \neq Operation(A) + Operation(B)$$

5.2 Datasets

For each operation we need one or more datasets which my be available on the same node that wants to run the operation initially or could reside on other nodes.

5.2.1 Input

Input files are normally not mission critical and could be reproduced.

5.2.2 Output

Operations create output datasets which normally are small in size, threfore we ignore the transfer cost of operation results in our work.

5.2.3 Data Locationing

We consider three different approaches toward preparing required data for operations.

Conventional Approach

in this approach we put the required data on a network file system and all application instances will access it there. We will utilize an NFS mounted file system.

Centralized Approach

in this approach we will have a central instance which will orchestrate operation delegation and operation output forwarding to other nodes.

Decentralized Approach

in this approach we will eliminate the orchestrator node and the network of application instances should collaborate in a decentralized fashion to keep track of data and control flow for each task.

For every approach we will run performance tests and we will compare the results.

Method

We will discuss scenarios in chapter5. For each scenario we will analyze the possible combinations of data and operations and we will discuss how to deliver the input data and where to store output data. We will discuss workflow management in chapter 6 and data transfer in chapter7.

5.3 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:

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.

5.4 Concrete Scenarios

We begin with a simple scenario and we gradually add details to it and build new scenarios.

5.4.1 Scenario 1

In this scenario we have a linear operation, e.g. Op^A on $Node^A$ which requires one single dataset such as $Dataset^1$ which is available on one of the other peers.

Conditions

$Dataset^1$ is not available on $Node^A$ and the operation is linear.

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 $Dataset^1$.

5.4.2 Scenario 1 (UC1)

We have a distributed network of collaborating servers, where in this case, we consider two computers. Each server has its own storage and maintains a number of datasets on it. These servers collaborate together to accomplish issued commands. User in this case wants to perform one operation on a dataset that resides only on one of the servers. There are two main assumptions here:

1. **The user has neighter a prior knowledge where the data is stored**
2. **Nor of how many servers are present on the network**

The user connects to one of the servers, which we call a client. This server is assumed to be part of the network, though it may not have any local data stores on it. The user issues, interactively (or non-interactively) a command on a set of data providing some kind of identification. This command is broadcasted by the client to all servers in the network. All servers receive this command and check whether they have the data locally. The server which has the data performs the operation and the others ignore it. The result of the operation in this case, remains on the same server which the original dataset was on.

- Note: it is assumed that at any instance of time, only one server acts as a client.

Moreover we assume the user has already queried the available data in the entire network by issueing something like "list datasets" which outputs dataset names and ids.

The following table shows two servers, each has one dataset. The user is connected to S2.

Server ID	*Dataset ID*	*Client*
S1	DS1	No
S2	DS2	Yes

Let us assume the data sets are 10^6 random numbers. Let us assume the operation is to transform the real random numbers to a set of [0 or 1] depending on whether the number is even or odd. This operation is assumed here to be a user defined method that operates on the data set.

- Note: A dataset can be for example defined as an object that has an id, and a one dimensional array (python list).

The user issues the command like this from a python shell:

- real2bin(DS1) will result in -¿ Broadcast(real2bin(DS1))
- Note: it is assumed that all functions are already defined on all servers, since they execute the same environment.

The client broadcasts this function to all servers. Each server will check if the data set with this id exists, if so will run the command.

This means that each server, especially the client, has to "know about all data sets existing in all servers. It does not need to have the actual data, but needs to know about it. So that when the user issues the command above, he/she does not get a "data structure not existent" error from the client, just because the data is not there. Hence we need some interface, or some wrapper function that checks the argument for the data type, or to create some proxy interface from all data to all nodes.

5.5 Assessing Suggested Approaches

To be able to assess the performance of each given solution to the mentioned scenarios we made a demo application called **Konsensus**. The code is available on Github.

5.5.1 Testing Problems

Writing tests for a distributed application is not as easy as writing unittests for a normal application. Our demo application acts as a server and client at the same time. Moreover we want to launch multiple network peers running on one or multiple machines. Testing scenarios on this network is not possible with normal mocking approaches, because we need to test the behaviour of our solution in a network of collaborating peers which are not external services, rather the core services of the application.

To overcome testing issues we have to launch the desired number of peers separately and then run our tests over them. To make this operation faster we changed the application to make it possible to launch any number of instances on one machine and we automated this process using a number of scripts.

Mixing Signals in Greenlets

We use python Greenlets instead of threads. This means that our demo application runs on only one thread. This causes a problem when launching

multiple apps all together with one script and inside one thread, that causes the signals for events spread among all greenlets and make trouble. To avoid this we have to run each server in a separate processes. Running them inside threads won't help as well because the blinker python library is threadsafe so it moves signals between threads as well as greenlets.

5.5.2 Scenario 2 (UC2)

This is similar to scenario one, except that the operation requires two datasets to be done. We have a network of peers collaborating to finish some linear and non-linear tasks. In this scenario we need at least three peers involved. We assume the first peer has no data of our interest therefore it should cooperate with others to accomplish the request. Our operation in this case requires two different datasets which are not available on the first peer and we should access them on other peers. The main points the same:

1. **The user has neighter a prior knowledge where the datasets are stored**
2. **Nor of how many servers are present on the network**
3. **The operation is linear**

We assume the data distribution is like the following table:

Server ID	*Dataset ID*	*Client*
S0	—	Yes
S1	DS1	No
S2	DS2	No

Possible Approach

In order to calculate the result we might take a number of approachers, we start with a combination of **divide and conquer** and **produce-consume-collect** methods.

The S0, in this case, is the peer who receives the command and initiates the request. The two other peers, S1 and S2 respectively, have the required datasets. The initiator will find the corresponding datasets and will dispatch commands to run each part on each peer and then will collect the resulting datasets. This will be a blocking operation, we will wait untill the other peers finish their parts and return the result to us. If the output is a number it will be returned to the user, if it is a dataset it will be stored based on defined storage mechanism, currently we use radnom storage. The peer will

break the operation into smaller operations each one calculating result for one of datasets, this **sub-operations** will be executed like **scenario 1** and the result will be collected by initiator peer.

- Note: in this case each pear should be able to run the requested linear operation on one or more datasets.

The notation of above mentioned approach will be like this:

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

In order to run this operation in a collective way, we need to think of the type of service calls in our system, whether they are blocking or non-blocking. Since often the operations in HPC environments are time consuming and long-running, we consider the non-blocking approach. In this way the user will provide a dataset name for storing the result. The operation will be **submitted** to the collaborative network. Later on user is able to query for the result using the key that she had provided at the time of submission. This allows us to design our system in a more decentralized way, where each peer can inform others (neighbors) about a request in a **publish-subsribe** manner, where the peer will publish a request and finish the operation. Later on the peer who has the dataset will **react** to the published request and will take further actions, all the other peers who do not have the requirents (the dataset for now) will ignore it, however they can store the details of running operations for next steps, when we will come to more complex workflows.

To show more detailed version of this operation we demonstrate the steps for it:

1. User issues the command to S0, providing DS1, DS2 ~~and a unique name for the result~~
2. System will check whether the operation is linear
3. Then it will break the command into sub-commands, each for one of datasets
4. System will generate unique ids for each sub-command
5. System will then submit the sub-commands along with dataset name and the unique id for the result dataset to **itself**, which will cause a situation like scenario 1
6. System will next have to collect the results in a non-blocking manner which we will discuss shortly.

- With the use of operation ids we eliminate the need to get a result dataset name from user but we still can accpte **tags** from users.

There is an important issue here, we create sub-operatiosn for each operation and we run them in a non-blocking manner, this will cause it almost impossible to return the result of operation to the user in one run. One might think that we can block and query until the result of sub-operations are ready, but this is something that we want to avoid. Therefore to solve this issue in a distributed manner, we introduce an operation id for each user request. We inform all the peers via sending messages (signals) about the new operation and it's id and sub-commands. Each peer will update this operation internally based on further received messages. We also return the operation id to the user instead of any results. Then user will query for the result of operation, providing the operation id. We change the above steps like this:

1. User issues the command to S0, providing DS1, DS2 and a unique name for the result
2. **System will generate a unique id for the operation and will store it along with the parameters**
3. System will check whether the operation is linear
4. Then it will break the command into sub-commands, each for one of datasets
5. System will generate unique ids for each sub-command
6. **System will notify other peers about the incoming operation with related parameters**
7. System will then submit the sub-commands along with dataset name and the unique id for the result dataset to **itself**, which will cause a situation like scenario 1
8. System will next have to collect the results in a non-blocking manner which we will discuss shortly.
9. System will return the operation id to the user

In the other hand the other peers which are the same basically, will react to the new operation signal:

1. Receive operation update message
2. Make a local lookup if the operation should be added or updated
3. Add or update the operation in the local storage

Having the operation id and local updating storage for operations we now need to find a way to collect the results. First of all we need to decide which peer will collect the results. We take the most straigh forward for now, the initiator peer, which has the knowledge of existing datasets in the network along with their sizes, will pick the peer which contains the largest dataset as the collector peer. We explicitly decide about the collector node in the beginning either by size or randomly amount the data container peers.

It is worthy to mention that the collector peer will then store the result based on the configured storage mechanism which is random storage for now, not necesserily storing on the same node.

Now we have enough information in each peer to collect, process and store the results. The peers (including the collector) will react to operation methods like this:

1. Receive operation update message
2. Make a local lookup if the operation should be added or updated
3. Add or update the operation in the local storage
4. Am I the collector? If yes do the followings:
    - check if the sub-operations are done
    - If the sub-operations are done, collect their results
    - Process the results
    - Based on the storage mechanism store the result
    - Update the operation with the result dataset id
    - Change status of operation to "done" (we need a proper state-machine here)
    - Inform other peers about the update

Now if user makes a query giving the operation id this would be the result:

1. Check operation storage
2. If the operation is marked done, return the dataset id
3. If it is not done, return the status.

Chapter 6

Workflow Management

Chapter 7

Data Transfer

7.1 Large Array Transfer

To transfer large arrays over the network there are a number of considerations. Should the array be stored locally before transfer? What if the array is so big that it does not fit into the machines memory? And how the array should be transferred?

Currenlty we assume the result datasets fit into memory, therefore there is only the question of how to transfer them over the network. To prevent unnecessary copies, we consider streams to send them to other peers. In the demo application this is done with streaming sockets. The other peer will be notified and then it will fetch the desired dataset.

7.1.1 Sequence Diagram

References

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

[2] Weiwei Chen and E. Deelman. "WorkflowSim: A toolkit for simulating scientific workflows in distributed environments". In: *E-Science (e-Science), 2012 IEEE 8th International Conference on*. Oct. 2012, pp. 1–8. DOI: 10.1109/eScience.2012.6404430.

[3] 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.

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

[5] 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.

[6] C. Moretti et al. "All-pairs: An abstraction for data-intensive cloud computing". In: *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*. Apr. 2008, pp. 1–11. DOI: 10.1109/IPDPS.2008.4536311.

[7] S. Pandey et al. "A Particle Swarm Optimization-Based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments". In: *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*. Apr. 2010, pp. 400–407. DOI: 10.1109/AINA.2010.31.

[8] K. Plankensteiner, R. Prodan, and T. Fahringer. “A New Fault Tolerance Heuristic for Scientific Workflows in Highly Distributed Environments Based on Resubmission Impact”. In: *e-Science, 2009. e-Science '09. Fifth IEEE International Conference on*. Dec. 2009, pp. 313–320. DOI: 10.1109/e-Science.2009.51.

[9] K. Ranganathan and I. Foster. “Decoupling computation and data scheduling in distributed data-intensive applications”. In: *High Performance Distributed Computing, 2002. HPDC-11 2002. Proceedings. 11th IEEE International Symposium on*. 2002, pp. 352–358. DOI: 10.1109/HPDC.2002.1029935.

[10] 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.

[11] S. Shumilov et al. “Distributed Scientific Workflow Management for Data-Intensive Applications”. In: *Future Trends of Distributed Computing Systems, 2008. FTDCS '08. 12th IEEE International Workshop on*. Oct. 2008, pp. 65–73. DOI: 10.1109/FTDCS.2008.39.

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

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

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

[15] Jianwu Wang et al. “A High-Level Distributed Execution Framework for Scientific Workflows”. In: *eScience, 2008. eScience '08. IEEE Fourth International Conference on*. Dec. 2008, pp. 634–639. DOI: 10.1109/eScience.2008.166.

[16] Qishi Wu et al. “Automation and management of scientific workflows in distributed network environments”. In: *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*. Apr. 2010, pp. 1–8. DOI: 10.1109/IPDPSW.2010.5470720.