A prototype implementation of Map/Reduce Model on Message-Oriented Middleware

Ioannis Polyzos
School of Engineering and Computing
Glasgow Caledonian University
Glasgow, United Kingdom
i.polyzos@acm.org

Huaglory Tianfield
School of Engineering and Computing
Glasgow Caledonian University
Glasgow, United Kingdom
h.tianfield@gcal.ac.uk

Abstract—This work studies current implementations of Map/Reduce Model and proposes an implementation shift for Inter-Process Communication mechanisms from currently used Remote Procedure Call (RPC) to Message-Oriented Middleware (MOM) in order to support the massive system scales in today distributed systems. A prototype is preliminary built up to demonstrate MOM implementation.

Index Terms—Map/Reduce, Remote Procedure Call, Inter-Process Communication, Message-Oriented Middleware

I. INTRODUCTION

The advances in networking technology in recent years, have led to an explosive growth of Internet and on-line services [1]. There is a trend to turn software into services [2] which are to be outsourced through the network as re-usable utility components according to the “Utility Computing” paradigm [3].

This shift to online services has emerged into models such as Software-as-a-Service, Platform-as-a-Service, Infrastructure-as-a-Service, etc. That very soon are either turned into Internet-scale services or to systems usable to support such services. The need for computational resources to support large-scale services is increasing considerably. This has led the research community to turn to Grid Computing. Grids aim to aggregate and share computational resources from clusters of hundreds or thousands machines. The use of Grids along with Virtualization to abstract the underlying low-level infrastructures has been the base to the recently emerging Cloud Computing [5], which is considered to be a paradigm shift to infrastructure and service delivery.

Processing of large-scale data is critical to many businesses in order to be able to respond effectively and quickly to opportunities of today where global markets are even more competitive than ever.

Even though the computations applied in data are not complex, the huge scales make the processing very difficult and time consuming. Thus Parallel Processing [8] and Grid [7] technologies are utilized to harness the computational power from clusters of hundreds or thousands machines. The use of Grids and Parallel processing introduces a new level of complexity mainly due to the distribution of codes and data as well as the fault tolerance capability of the system [7], [8].

As a tackling to this complexity several methods have been introduced. A recent model for parallel processing, Map/Reduce [6], has gained high popularity and wide spread in the community. Map/Reduce was originally undertaken in Google, to effectively parallelize computations of large scales of data such as crawled documents and web request logs. It has extensively been used inside the company along with other systems developed in house such as Google File System to support its famous search engine and other systems with great success.

Map/Reduce has been used widely in academia, open source and industry to solve problems in a wide variety of domains. Important factors to the popularity Map/Reduce enjoys in the community is its simple and straightforward yet powerful design, as well as Hadoop [9]. Hadoop is an open source clone of the system presented by Google and is considered today as one of the most mature implementations. Initially started from Doug Cutting and now is hosted under the Apache Software Foundation with a large community to support its development.

There are several implementations of Map/Reduce that vary in different ways. Many companies have built their own private implementations while several open source projects have also been initiated. Among the projects developed in the community some interesting implementations are: CouchDB [10] a document oriented database which makes use of Map/Reduce to define views and distribute documents, Disco [11] a implementation in Erlang, FileMap [12] a lightweight language-neutral Unix-Style implementation, Misco [13] a Python based implementation targeted at mobile devices and Mars [14] an implementation targeted at Nvidia GPU graphic processors.

Map/Reduce performs well under heavy loads. However, when systems grow massive in size, Scalability, Reliability and Fault Tolerance of distributed systems will be affected. This would be challenges for Map/Reduce model implementations. Systems are getting hard to support when, for the purpose of increasing performance, an increasing number of resources is added for the system to operate upon.

This work is focused on Map/Reduce model of the underlying system in implementations based on Google’s system and Hadoop. The Inter-Process Communication (IPC) mechanisms are explored and an alternative approach is presented using MOM implementation. A prototype is developed to demon-
strate this alternative approach. Finally a discussion follows to present several views of the new system.

II. RELATED WORK

A. Remote Procedure Call (RPC)

Most of Map/Reduce implementations make use of Remote Procedure Call (RPC) for Inter-Process Communication (IPC) Middleware.

RPC was introduced in early 80’s by Birrell and Nelson [24], [25] as an extension to the standard mechanisms provided by most imperative non-distributed languages for IPC. RPC uses a client/server [37] architecture model and extends this mechanism in order to support calls on procedures which belong to different processes that reside in machines across the network [19].

A collection of procedures that is allowed for a client to call is specified through interfaces. Such interfaces act as contracts between client and server. Interfaces are specified by means of an Interface Definition Language (IDL) which is subsequently compiled into client and server stubs. Invoking a remote procedure uses these stubs to generate calls to the methods as if they were local. A local call to the stub is marshaled and transferred to the server where the execution continues. The execution on the client stops until a response is received from the server. After the server sends back the response the client un-marshals its calls and the execution flow continues.

RPC, being simple and straightforward, has become very popular paradigm for developing distributed applications and has been widely used for building various systems both in academia and industry with great success [20]. However early RPC implementations such as ONC-RPC [26]/XDR [27] which is subsequently compiled into client and server stubs. Invoking a remote procedure uses these stubs to generate calls to the methods as if they were local. A local call to the stub is marshaled and transferred to the server where the execution continues. The execution on the client stops until a response is received from the server. After the server sends back the response the client un-marshals its calls and the execution flow continues.

RPC systems are tightly coupled to specific interfaces and argument types that both should be implemented by the systems.

-Synchronous nature of RPC systems makes the code blocked until a response is made available.

-Failures in RPC systems may lead to erroneous responses which may affect data integrity, or execution delays due to timeout mechanisms.

-Data marshaling formats differ between implementations.

This restricts systems interoperability.

B. SOAP and WSDL

Innovative work of XML-RPC [21] encodes transmission of RPC in standardized way with XML (extensible mark-up language). The use of XML eliminates the use of stubs while it provides users with primitives to create their method requests and responses and improves interoperability and marshaling as data are encoded in text before sending.

Important feature of XML-RPC is that it could be layered easily on top of existing network application protocols such as HTTP which is the basis of Web Services.

Today Web Services commonly make use of XML or SOAP [28] - a W3C recommendation protocol for exchanging information replacing XML-RPC (which is currently halted).

SOAP-RPC systems also use interface compiler along with Web Services Description Language (WSDL) [29] - an XML formatted language for IDL that acts as a contract between client and server. WSDL is a standard that improves upon interoperability of RPC and is compatible with protocols other than SOAP.

Though Web Services using SOAP, WSDL and other standards constitute a great evolution to early RPC implementations, there are still issues related to underlying transports such as HTTP. HTTP due to inherited unreliability of its underlying protocols makes it inappropriate for systems where guaranteed delivery and other Quality of Service (QoS) are required [30], [31].

C. Message-Oriented Middleware

To deal with reliability issues, reliable transports have been introduced such as HTTPR [31]. Also specifications such as WS-Reliability [32] and WS-ReliableMessaging [33] have been developed while many products (open source and proprietary) offer non standard (not part of WS-I) transport using Message-Oriented Middleware (MOM) and JMS [34]. The use of MOM and JMS as a communication transport is often referred to as "SOAP over JMS".

Message-Oriented Middleware [36] refers to infrastructures and common approaches used to provide means of communications between distributed systems. In MOM communication between systems is realized in the form of exchanging (sending and receiving) messages from one application to the other across a network.

The main idea of MOM is based on Message Queuing [35] and message mediators - infrastructures that stand between senders and receivers of messages. Mediators are also referred to as “Message Brokers” and contain virtual channels called destinations where messages are sent to and delivered to receivers.

Since a message is sent, it is first delivered to a destination and not to the actual receiver. The message received to the destination may later be sent to the applications registered or subscribed for interest on this destination. Messages delivery patterns may vary, common supported models of which are typically those of: point-to-point (p2p) and publish/subscribe (pub/sub).

This mechanism allows for loose coupling between the message senders and message receivers because this process is taking place asynchronously without requiring for each to be available or knowing anything for the others. MOM with additional work has made it possible to support non tight-coupled RPC systems [36]. Some of the most important features include message persistence, message routing and message transformation.
Vendors of MOM provide developers with specific API for sending and receiving messages as well as utilize specific features. However there are open, vendor-agnostic API such as Java Message Service (JMS) that can be used with many different MOM vendors.

III. INTRODUCTION TO MAP/REDUCE

While discussing on Map/Reduce one may refer to either the Map/Reduce model or to the underlying infrastructure used for the model to execute on.

A. Map and Reduce Functions

Map/Reduce model, was introduced by Google to effectively parallelize computation on large-scale processing. Map/Reduce is built according to divide and conquer - a fundamental computer science concept. The basic idea is to partition large problems into small ones in such a way that the partitioned sub-problems are independent of each other and can be processed in parallel from different systems. Using such an approach in a distributed environment introduces several issues that need to be carefully thought of e.g

(i) How to break a large problem into small ones.
(ii) How to assign computation tasks to distributed processes.
(iii) How to coordinate distributed processes.
(iv) How to ensure data delivery.
(v) How to share results.

Map/Reduce model has its roots in functional programming and languages such as Lisp, Scheme and ML using features such as high-order functions while it takes advantage of side-effect-free nature of these languages and data-immutability.

Map/Reduce model is based on two higher order functions which are commonly built-in to functional languages: Map and Reduce.

Map: takes as argument a function M and applies it to all elements of a given list. Applying computations on the elements Map function results in new values.

Reduce: take as argument a combining function R e.g. , addition and applies it cumulatively to all elements of a given list to reduce them to a single value

Applying Map functions to the elements of a list, is a natural fit to parallel programming as function is applied on data in isolation, while in a distributed environment this function may run in different machines. The same applies also in the case of Reduce while this is more restricted as data need to be grouped first.

Using these two simple functions a programmer can define computations and algorithms to apply in large-scale data sets while they are automatically distributed across the cluster. Details of data and code distribution, aggregation of results, coordination and synchronization of execution as well as fault tolerance are totally abstracted and left on the infrastructure implementation.

A Map/Reduce program is often referred to as Job and in common Map and Reduce functions contains code along with configuration parameters

B. Mapper and Reducer Signatures

Map and Reduce functions may be implemented based on the following signatures:

map (k1,v1) - list(k2,v2)
reduce (k2,list(v2)) - list(v2)

C. Execution

In a Map/Reduce application there are two main system roles: Master and Worker.

Workers: are processes assigned to Map or Reduce tasks.
Master: is a special copy of the program responsible to check workers availability and to coordinate the execution.

Execution sequence of Map/Reduce programs can be summarized in the following steps:

Step 1. The user application with the use of the library splits input data into M map tasks.
Step 2. "Master" assigns map tasks for processing to idle worker processes.
Step 3. The worker assigned to a Map task applies the function in the contents of the split resulting in intermediate key/value pairs (Reduce tasks).
Step 4. Reduce tasks are buffered into memory and are periodically written to the disk and information about their location later is passed back to the Master.
Step 5. When location of data is fed back to the Master, data are read with the use of RPC and are assigned Reduce tasks to idle workers that read all data grouping and sorting them based on intermediate keys.
Step 6. Reduce workers iterate over sorted keys and apply upon the set of each intermediate key. The output of this process is appended to a final output for the specific partition.

Upon completion of the Map and Reduce tasks the system wakes the user program and R output files (one per Reduce task/intermediate key) are made available.

Figure 1 present the execution and data flow between Map and Reduce functions as described above.
D. Synchronization

Synchronization in Map/Reduce is oversimplified by use of data-immutability and side-effect-free properties the model inherits from functional languages.

Data result from Map and Reduce tasks are new data based on previous given input, eliminating the use of shared memory and the associated issues, which acts as a logical barrier between the Map and Reduce phases like Shared Nothing architectures [18].

E. Fault-tolerance

A critical issue related to the distributed nature of the system is that participant systems eventually fail [16] either due to the presence of hardware or software errors. Map/Reduce library presented by Google tolerates gracefully failures of both worker and master machines.

Master nodes periodically ping workers to check their availability. Whenever workers do not respond they are marked as failed and tasks assigned to them are reset in order to be assigned in other workers available. Master nodes also write periodic checkpoints of their data structures in order to be able to continue Master tasks in case of failure.

IV. PROPOSED DESIGN

Using Message-Oriented Middleware will lead to loosely-coupled and more reliable systems than using RPC-style invocations [39] SOAP over JMS [34] has been a strong example of this, which has shown the advantages of JMS over RPC in Reliability. This is an inspiration for our work.

Implementation of Map/Reduce using MOM is considered to be a promising work as many tedious issues of RPC are resolved while it is leading to loosely coupled distributed systems able to support massive growth of systems size and tolerate the presence of errors.

Besides Reliability as in “SOAP over JMS”, Fault-tolerance and Scalability are also improved with the use of MOM while a slight separation of concerns is realized in terms of components and responsibilities.

In this section we design a prototype Map/Reduce implementation based on MOM is presented.

A. Model

Our Proposed implementation does not affect the Map/Reduce model. In addition, interfaces remain almost the same in order to eliminate the changes and ease experienced programmers to work with the framework.

Parts that are considerably affected belong to the underlying implementation that the model operate on, including data types and structures as well as the execution flow which can be seen below is actually simplified.

B. Execution

Execution flow of our prototype system remains very similar to the flow of Map/Reduce implementation. However, some of the tasks previously implemented in the framework are now being performed by Message-Brokers using functionality common in brokers compliant with the JMS specification.

The flow followed in our prototype implementation is:

(i) The use of the library in a user application splits input data into M map pieces submitted later to Map Queue.
(ii) Workers pool messages from Map queue and apply Map function upon the data. This operation results in intermediate data being submitted to Reduce queue.
(iii) Workers retrieve messages from Reduce queue and apply Reduce function upon them. This operation results in R output files, one for each unique key.

An overview of the execution flow is depicted in Figure 2.

C. Data Types and Structure

Data and information exchanged in the system are of the following three categories:

(i) Information and Meta-data related to the Job.
(ii) Code to be used to process the data inspired from the code as data idea in functional languages.
(iii) The actual data to be processed and output of processing phases (Map and Reduce).

The input data, intermediate values and outputs are exchanged between systems in the form of message. Information and meta-data (such as parameters) about the Job along with the code used for the processing are wrapped in the Job Object. Job Object contains information to uniquely identify a job and meta-data information such as properties used by the run-time, as shown in Figure 3.

| UII | Name | Mapper Code | Reducer Code | Parameters |

Fig. 3. Job Object structure.

Information contained in Job Object is critical to all parts of the applications flow. In order to reduce multiple interactions between system components and decrease interdependence
Job Objects have made a part of message exchanges. Their comparatively small size seems to make this option ideal while enabling code mobility.

Data are exchanged between the systems in the form of messages. Messages, as Job Objects are modeled as simple arrays. They contain both the data and the code used to process the data. This makes the systems less interdependent while leveraging workers for history and state records.

The messages used to exchange information are such as MapMessage and ReduceMessage (these messages may later also be referred as tasks). In order to represent the array in the simplest possible form Map message types are used as provided by the JMS API.

Map-Messages are produced in user application upon the split of data with the use of the provided library. Map-messages are usually big in size and contain three parts: Unique Map Identifier (UMI), Job Object and Data, as depicted in Figure 4.

ReduceMessages are the result of the Map function application on the data splits the so-called intermediate values. ReduceMessages are usually small in size as they represent a form of processing output and contain four parts: Unique Reduce Identifier (URI), Job Object, Intermediate Key (iKey) and Data (intermediate value), as depicted in Figure 5.

V. DISCUSSION

Changing the IPC mechanisms on Map/Reduce introduces minimal or no changes to the programming interfaces. However, though abstracted, there are several changes introduced on the underlying mechanisms of the system following changes in Reliability, Availability, and Fault-tolerance as well as the interactions and coupling between the systems.

Designing and building systems using MOM in the IPC layer leads to more loosely-coupled and more reliable systems comparing to that using RPC-style [39].

Loose coupling is mainly achieved with the use of Mediators. Mediators stand between interacting processes. Messages are sent to a destination, first stored in the disk and then forwarded to recipients. Messages remain persistent until an acknowledgement of successful processing is received. This applies both to Map Messages and Reduce Messages, and increases reliability of communication and guarantee of message delivery as in "SOAP over JMS" [34] systems.

Worker systems are set free from persistence related tasks that previously were with RPC style. Setting workers free of persistence and reliability related tasks leads to a separation of concerns in terms of component responsibilities.

Message-broker in this new design takes the role of the Master component responsible for both reliability and coordination tasks.

Coordination tasks are of minimal complexity due to the simplified interactions introduced with the use of MOM. User applications interact with the broker to send the data splits, workers consume data splits and send back intermediate values, and finally reducers operate on intermediate values and produce output files.

Coordination tasks are closely related to fault-tolerance, grouping and ordering of messages. Grouping and ordering of messages is common to be provided by popular brokers compliant with the JMS specifications such as those of ActiveMQ, RabbitMQ and others.

Fault-tolerance semantics, conceptually and at a high level remains the same with the implementation of Map/Reduce introduced by Google. However, this is now a responsibility of the Mediators (brokers) and is commonly provided by most of the broker vendor implementations. If a worker has a considerable delay to send back acknowledgement of successful completion of a task the broker considers this execution as failed and the message is re-queued in order to be executed from another worker.

As previously discussed all messages sent to the broker are first persistent which ensures that no data may be lost in case the system fails. All data are available to the system since it restarts and execution may continue normally. This is similar to periodical writes that master components make on the implementation proposed from Google. However, similar to the case of persistence these are not required to be implemented by the Map/Reduce implementation but are guaranteed by the broker vendors. Also they contain all data usable for processing and set tasks with no dependencies to other systems such as DFS.

Brokers similar to master components in RPC style implementations are central to the system, which is not considered to be a good practice as they are focused points of failures. In case a broker fails, even if the data are persisted and secure, communication of worker components with the system is not possible. In such a case, workers will re-try to connect with the broker several times in specific intervals. After the broker is fixed and available the flow of execution may continue as if no problems happened.

Another issue with brokers due to being central is performance. Due to large amounts of data exchanged through the broker and the increasing size system that grows massive over time, the system at some point is unable to perform.

To overcome these shortcomings a cluster with more than one mediator may be used in a master-slave setup. Messages sent to brokers are either replicated between the brokers or a Distributed File System is used to replicate storage.

Workers using a simple Round Robin scheduling [40] could balance loads between brokers. In addition to load balancing, using a cluster of brokers is also beneficial in case of broker failures. Since the system operates in a cluster of brokers, in case of a broker failure, workers can use other available brokers in order to complete their tasks.
The use of a cluster of brokers in addition to the asynchronous model of MOM and the high decoupling introduce changes on the availability mechanisms on the system. The system is tolerant to unavailability of both workers and brokers. In case many worker processes fail or become unavailable the rest of processes and brokers are not affected. They can continue operate as normal.

Even in cases where all workers made unavailable the data are kept persistent and after workers become available the execution flow may continue as normal. Similarity this applies also to brokers, except the rare case where all brokers become unavailable. In the case of brokers unavailability work load is gathered in the available brokers and in case of many broker failures, issues in performance may arise.

Though many areas have gained significant benefits, there have been also disadvantages that may arise on the current design. In Hadoop and Google implementations, data exchanged between systems are written and read from Distributed File Systems (DFS). Instead of transferring the data between the systems, only information on their locations is shared. Every participant system then is able to retrieve these data using the DFS. Similar use of DFS has also been discussed earlier for sharing storage in the case of using a cluster of brokers.

In our prototype, as previously discussed, data are transferred between systems with the use of Messages through the network. Data delivered in the form of Messages and travel through the network give an additional level of decoupling in the application and decrease interdependence of systems. Additionally the system is decoupled from any DFS which may not be usable or desirable in specific environments.

However considering that Map/Reduce is used to process large-amounts of data there are consequences from the performance of the network due to large-amounts of data that are usually need to be transferred. Future work on the system may be toward this issue by additionally enabling the system to support transfer of data location information instead of the data itself and the use of DFS. Working in two modes the system may increase flexibility and its areas of application.

VI. CONCLUSION

In this work we have discussed the new challenges that may rise to distributed systems due to the massive system scales in order to support the ever increasing demands of today. Map/Reduce - a system designed to automatically distribute computation on large-scale data and harness computational power of large clusters, have been analyzed and current implementations have been evaluated.

Analysis and evaluation focus on the Inter-Process Communication (IPC) layer. Limitations of Remote Procedure Call (RPC) mechanisms currently used have been evaluated and the use of Message-Oriented Middleware has been proposed as an alternative IPC in order to be able to address this massive system scale.

Analysis and design of a prototype system built to support this work have been presented and a discussion follows on the major areas affected by this shift in IPC.

REFERENCES


[30] Reliability of SOAP over HTTP Web Services
http://myarch.com/reliability-of-soap-over-http-web-services


[38] Java Messaging Service (JMS ), http://java.sun.com/products/jms
