Overview of the Course
Why Compute in a Distributed Environment?

Distributed Computing

Definition: Distributed Computing is an approach to computation in which multiple (independent) computers or computing systems work on solving the same problem within the same time frame[1].

Distributed computing is facilitated via the means of:

- Distributed systems: collections of computers connected to each other via a network running coordinated software.
- Distributed computing frameworks: implementations of programming models that provide programmable interface to computing in a distributed way, but abstract away the distributed system organization details.

In this course: we are interested in understanding how to use ***existing distributed computing systems and frameworks*** to build **efficient solutions** to typical problems that require distributed computing.

In CSC 469: you will learn how distributed systems are organized, and will build a variety of them.

What this Means for the Course

1. We will largely view distributed systems and distributed computation frameworks as **black boxes**.

[1] This is a slightly rephrased definition borrowed from Techpedia (https://www.techopedia.com/definition/7/distributed-computing-system).

2. We will **not** address the issues of distributed system design and architecuture.

3. We will, *for the most part*, ignore the following eight Fallacies of distributed computing [1]:

   (a) The network is reliable
   (b) Latency is zero
   (c) Bandwidth is infinite
   (d) The network is secure
   (e) Topolgy does not change
   (f) There is one administrator
   (g) Transport cost is zero
   (h) The network is homogenous

   (by "ignore" I mean that if we need to assume any of these as part of our problem-solving, we will do so. It is the task of the underlying implementation of the distributed system/framework to deal with these issues)

4. While this is a somewhat superfulous distinction, this is an "applications development" course, not a "systems" course.

Why Distributed Computing?

Question: With the amount of processing power available to use on a single computing system, why do we need to build distributed systems and facilitate distributed computing?

There are multiple answers to this question. The driving force of this course is the following simple answer:

Big Data

For our purposes, we give the following definition:

Definition. We define Big Data as any data collection *that is larger than the storage capacity of a single computer system on which it is to be processed.*

Data-centric View of Distributed Computing.

In this course we take a data-centric view of distributed computing. This means that:

1. We view data processing (acquisition, managment, analysis, integration, transformation, exchange, visualization) as the chief purpose (*raison d'être*) for distributed systems.

2. We largely consider distributed computing solutions for "pure" data processing problems in the course.

The role of Big Data

The data-centric view of distributed computing explains why Big Data (defined as above) is the answer to the **"Why?"** question:

- A data collection that cannot fit the storage capacity of a single computer system must be *distributed across multiple independent systems.*
- There is *no shared memory*[2] between multiple indpendent systems (as inferred from the use of the term "independent").
- Individual systems storing the data collection will have immediate access to *portions* of the entire data collection.
- Therefore, any computational task involving the data collection *may need to be executed on multiple systems.*

So, what can we learn?

In this course we want to address the following two questions:

What common problems need to be solved in a distributed environment? As stated above, we are concentrating largely on *data processing* problems. Some of the types of such problems are:

1. Data indexing and retrieval (search) problems.
2. Data aggregation problems.
3. Data organization/partition problems.
4. Data integration/joining problems.
5. Data mining/machine learning.

How to solve these common problems using existing distributed computing frameworks and distributed systems? In this course, we will cover a number of frameworks that allow for distributed data processing:

1. Key-value stores. Sometimes referred to as NoSQL database managment systems[3], key-value stores are distributed computation engines that evolved in the past few years to address the problems of indexing and retrieval of very large data collections.

[2] Although, as we will see in this course, there may be shared storage.

[3] Technically, key-value stores, and document stores that we will be looking at in this class are a subset of all NoSQL DBMS.

2. MapReduce framework. The MapReduce distributed computation framework partitions a data processing task into smaller steps that transform data (a Map step) and aggregate it (a Reduce step). A MapReduce computation framework abstracts the distributed nature of computations and allows developers to concentrate on the data manipulations instead.

3. MapReduce extensions. MapReduce comes only with two types of data manipulation procedures, a Map-style transformation procedure, and a Reduce-style aggregation procedure. Some types of tasks may naturally break down into components that are not a mapper or a reducer. MapReduce extension frameworks add other types of user-created distributed data manipulation procedures.

4. MapReduce add-ons. These are higher-level frameworks built on top of a MapReduce framework. These frameworks allow the user to express their data processing needs in a declarative form, and they automatically translate such declarative expressions into a MapReduce job.

In the class we will use specific *embodiments* for each category of distributed computation framework/system, but our overall stress is not on learning how to deal with a specific implementation, but rather, learn the general prinicples that are followed by all such implementations.

In the current offering of the course we will use

1. MongoDB as an example of a key-value and document store.

2. Hadoop as an example of a MapReduce framework.

3. Spark (if we get there) as an example of a MapReduce extension framework.

4. Pig or JAQL as an example of a MapReduce add-on. (We might also look at Hive just a little bit, but what Hive does is a bit outside of the algorithmic content of this course).

How are Distributed Computations Different?

Even ignoring the fallacies of distributed computing, in order to create a distributed solution, one needs deal with a lot of issues.

1. How to store data on multiple systems? What are the ways to partition the data? Should there be any data replication?

2. How should one part of a distributed system communicate to other parts?

3. How does one ensure consistency of data? How are data updates handled?

4. Each compute node in the distributed system only sees a part of the data. How does one assemble the full answer?

Any distributed computing framework must deal with these issues. Well-designed frameworks abstract these issues for the developer.

But What About Databases?

There is one area of Computer Science that deals with data processing and management: Database Systems.

Distributed Relational DBMS exist.

In fact, all major Relational DBMS can be set up in a distributed environment.

Question: How does this fit into this course?

Answer: We leave the study of Relational DBMS to CSC 365. Here are the reasons:

- Distributed RDBMS differ from single-server RDBMS (or single-server RDBMS configurations) at the system organization level - a distributed RDBMS requires new components responsible for data distribution and communication between individual servers. *However*, the key means of communicating with a Distributed RDBMS is **exactly the same** as teh means of communicating with a single-server RDBMS: SQL.
- CSC 365 already covers the relational data model and SQL as part of the course. Since these do not change, there is no need for us to address these issues in CSC 369.

Having said this, declarative querying of data in the style of SQL is a popular feature for distributed systems. We may wind up looking at declarative query languages that borrow a lot of SQL syntax. Knowledge of SQL will help, but is not necessary to learn. Conversly, if you do take CSC 365 later, you will be able to transfer some of your CSC 369 knowledge.

References

[1] Arnon Rotem-Gal-Oz, Fallacies of Distributed Computing Explained, *white paper*, http://www.rgoarchitects.com/Files/fallacies.pdf.