**Tribhuvan University**

Institute of Engineering

**Pulchowk Campus**

Distributed System

---

# Assignment2

---

Submitted by:

Bishal Katuwal

075BCT028

Submitted on:

21st June, 2022

# 1 Assignment 2: Short notes on Distributed Debugging

**Bugs in distributed system**

For a single machine systems, OS has global view and timestamp logs makes sense. But when dealing with distributed systems, OS mo longer has global view and has limited control over system. The operation not only depends on state and timestamp of one machine but also on state and timestamp of another machine. Thus debugging a distributed system is challenging. In a distributed system, a program is composed of multiple independent but interconnected computing nodes. The nodes exchange messages to coordinate the distributed computing. Regardless of whether the individual nodes use concurrency, the overall distributed application is automatically concurrent because each node has its own execution path.
This means there is no way to know the order in which the program paths execute. Although a distributed system can be simulated and debugged in development phase, It is impossible to recreate the exact environment where the system will be deployed. Thus not all issues are solved in development phase. Furthermore, it includes both the concurrency as well as the distributed execution as a source of errors.

**Distributed Debuggers**

Most errors can be solved by reading the code and reading the error messages. However, it may not be enough. This gives rise to proper debugging methods. Some traditional debugging methods are:

- Ad hoc method( using print statements)
- Code checkers
- Unit tests, integration tests and end-to-end tests
- Bisecting
- Logging
- Profiling

Again, the complexity is higher for distributed systems. Thus these methods do not completely fix all bugs. The following are some techniques which can help with debugging distributed systems.

1. Contracts and documentation
   The easiest way that can be implemented by developers for distributed debugging is specifying message protocol between nodes. Since it is impossible to debug a distributed system without proper understanding of communication between and among nodes, proper documents and contracts are important.

2. Defensive Programming
   Another method for programmer is called defensive programming. After document contracts have been set up, the system should give an error or warning signs whenever the contract is violated. This makes it easier to locate bugs in early phase of development and handle them.

3. Better tests
   A better way to debug any system is to simply have better testing methods. For distibuted systems, a better way of testing consists of two steps. Ensuring that operations within a node work as intended followed by end-to-end tests for communication among nodes.

4. Remote debugging
   A locally running debugger can be connected to a remote node of the distributed system. This allows the use of the same features as debugging a locally running program. Key problem here is to identify which node to connect to and to avoid network timeouts while debugging parts of the distributed system.

5. Distributed Logging
   Log collection tools (e.g. Prometheus or Logstash) collect and transform log files, then insert them into a data store (e.g. Prometheus or Elasticsearch). Visualization tools (e.g. Prometheus or Kibana) allow developers to query and visualize the data across all nodes of a distributed system or selectively for certain nodes. This can be very helpful when looking for errors messages or certain outputs. Also, distributed systems usually emit metrics which can be an indicator for what is happening during execution. Apart from application specific metrics, standard metrics like CPU usage, memory usage, and network saturation are typically logged and available. This also servers as a tool to locate any bugs.

6. Model checking
   By building a model of the concurrent and distributed aspects of the systems, we can formally specify important aspects of it. Further, we can then run model checks to see if the system is guaranteed to run correctly, according to our model.

Debugging distributed systems is hard, but not impossible. With the right tools and practices it is a reasonable endeavour. Extra care must be taken when designing a debugging tool as it mustn't itself introduce inconsistent state and hide bugs. A debugging tool must also not be resource heavy as the cost must be kept in check.