ASSIGNMENT # 1

Simulation of Distributed System

Abstract

This assignment is based on principles of clock consistency, associated clock drift, inter-process communication, security of data over un-secure communication medium in distributed system and design decisions which are influenced by end to end argument. In this assignment I have implemented Lamport's logical clock which is based on Berkeley's algorithm. In this assignment I have proposed interaction model and failure mode along with its pros and cons. Moreover, I have discussed design decisions which are influenced by E2E argument and, I have performed encryption/decryption of message exchange between master and process object.

Pawar, Rohit Sidram
rspawar@iu.edu

Interaction Model

Since this assignment is based on socket programming, any two processes running on different processors are communicating with each other using message passing so point to point interaction model is used. In this simulation, I have created one master object and four process objects, and they are communicating with each other using message passing. Process object is sending clock value to the master object in case of send event and requesting for an offset value for receive event. In this model each process has its own clock value and it will increment on every event.

Pros of the design:

1) By using this model, we can simulate distributed system in which each process object can run on different machine parallelly and they can communicate to master object using message passing.
2) Since the application is distributed in nature, failure of one process will not affect any other process execution.
3) Because of this simulation process object can perform event ordering by communicating to master object.
4) Every process object can be used to perform specialized functionality.

Cons of the design:

1) Since the application is distributed in nature and we have implemented notion of logical clock hence global clock is not possible.
2) Process objects are not aware of other process object clock value and they can change the clock value of other processes.

There are two types interaction model:

1) **Synchronous model**: It has assumption about time to perform some operation, communication time due to which it has disadvantages and it's very difficult to calculate realistic value.

2) **Asynchronous model**: It has no assumption about time. e.g Internet

I have developed a system using asynchronous interaction model, in which there is no time bound associated with communication and different operations. In my model:

1) Each process has its own execution speed.
2) Message transmission time between any process object and master object is different.
3) Clock drift is also unknown.
4) Each process has different probability for different events.

Because of asynchronous model, implementation of simulation is very simple and important reason to choose this model is process object doesn't get blocked.

In my simulation, I have performed event ordering using notion of Lamport's logical clock which is based on Berkeley's algorithm.

Pros of my design (using Lamport's clock based on Berkeley's algorithm):

1) No need of global clock, events can be ordered using logical clock.
2) We can implement logical clock just by using simple counter, which will increment upon every event.
3) Implementation is simple because every process object is communicating to only master object.

Cons of my design (using Lamport's clock based on Berkeley's algorithm):

1) Since the design is based on Lamport's logical clock so drift in one clock would cause drift in all other process object and clock drift has known limitation.

Failure Model

1) **Omission failure:**
   Process object and Master object, both can detect crash of each other's by catching appropriate exception. Moreover, we can detect crash by using timeouts provided guaranteed delivery of messages, but it's not implemented in current system.

2) **Arbitrary failure:**
   Each client has different probability to generate arbitrary failures. Client request for an offset value from server and just ignores received value by not updating its own clock value by an offset.

Pros of my design:

1) Process and Master object, both can detect crash of each other.
2) Failure in any process object doesn't stop execution of application as other process object can continue its execution so I can say due to which I am able to achieve some sort of inherent distributed nature for my application.
3) Process object can detect crash of master object and terminate its execution in proper manner.

Cons of my design:

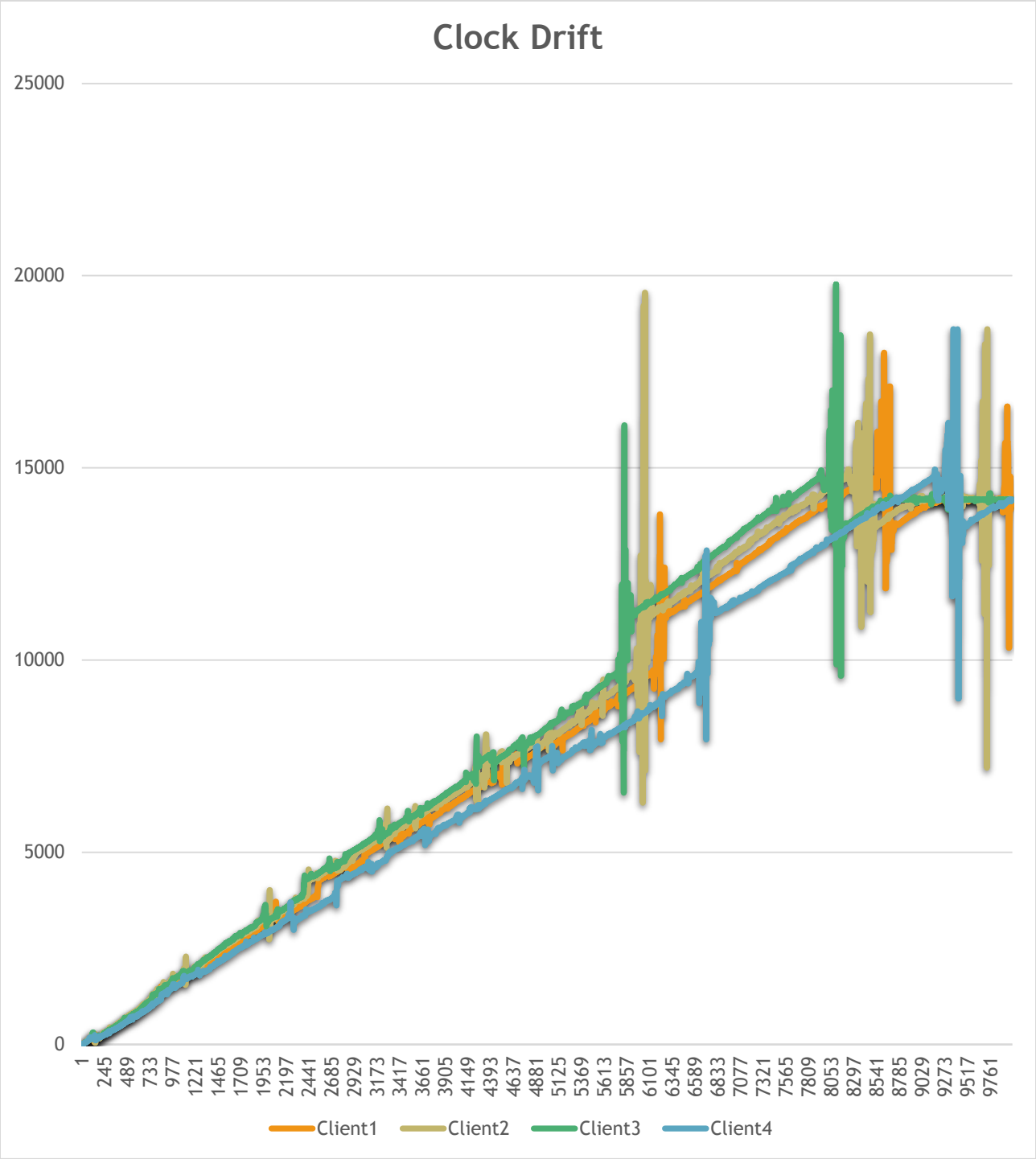
1) We can detect crash by using timeouts/heartbeat provided guaranteed delivery of messages, but it's not implemented in current system.

Discussion about results using Graph

I ran 4 process objects for 10000 iterations with different probability as mentioned in below table and generated two graphs of 10000 iterations and 1000 iterations for better understanding:

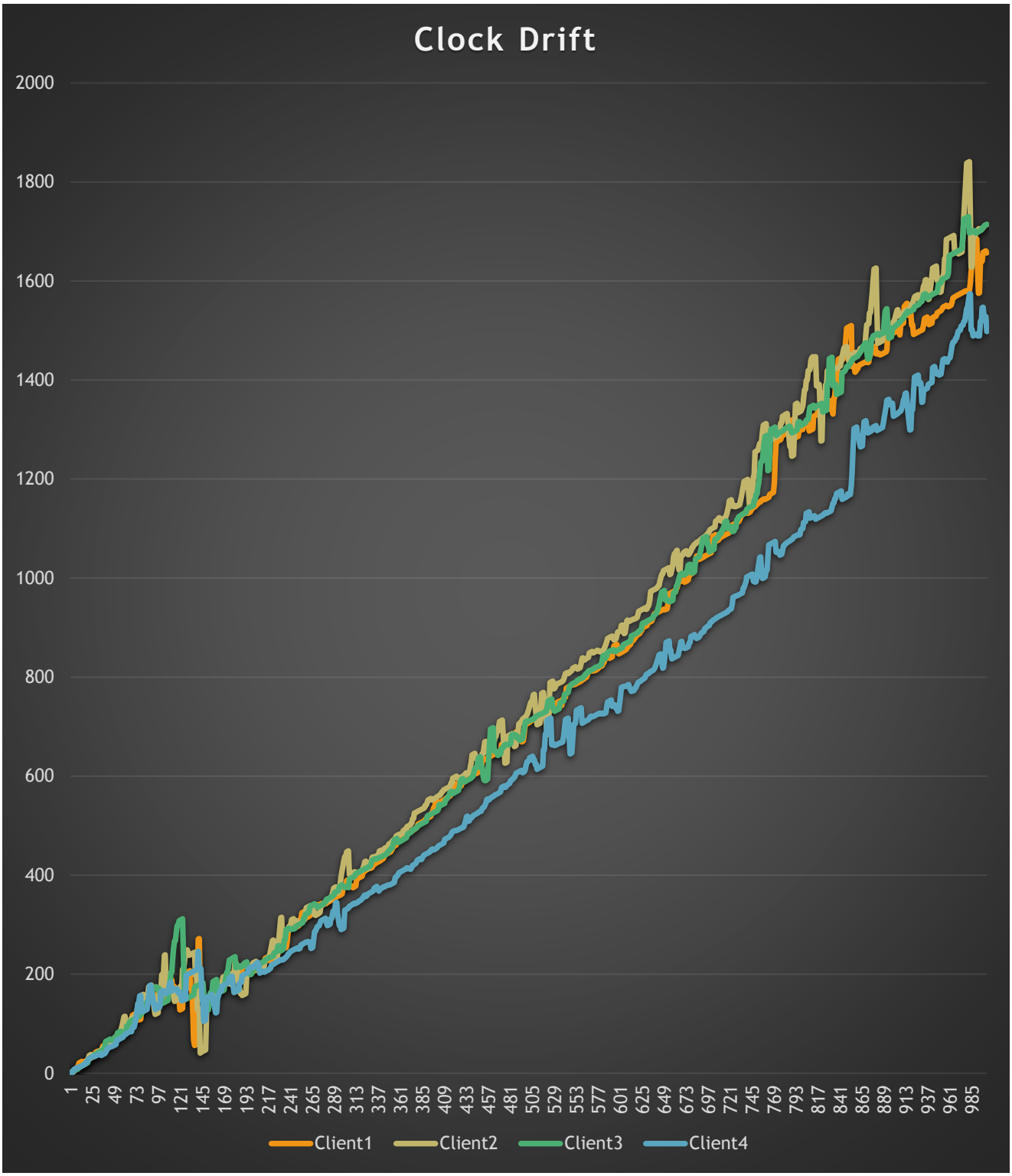
No.	Process Object	Send	Receive	Internal Event	Byzantine Failure
1	Process Object 1	0.33	0.33	0.20	0.04
2	Process Object 2	0.40	0.30	0.25	0.05
3	Process Object 3	0.35	0.35	0.19	0.11
4	Process Object 4	0.30	0.45	0.22	0.03

Graph1: It consists of data of 4 process objects with 10000 iterations. At any iteration we can see clocks are almost close to each other hence we can say that, process objects are almost in sync. However, at some instances little bit spike is observed, that's because different process objects have different event probability which includes byzantine behavior also and clock drift has known bound.



Graph1: Graph of 4 process objects with 10000 iterations

Graph2: It consists of data of 4 process objects with 1000 iterations for better understanding. At any iteration we can see clocks are almost close to each other hence we can say they are almost in sync.



Graph2: Graph of 4 process objects with 1000 iterations

Design decision taken with respect to E2E principle

1) **Encryption/Decryption at ends:**
   Communication sub-system is designed in simple manner for better performance. We cannot completely rely on communication sub-system hence I have implemented encryption /decryption at end points as ends have good knowledge about application. If encryption/decryption is not implemented at ends, then I will have to secure entire communication sub-system then it would have cost more as well as it would be very complex sub-system hence other applications which are using it will have to pay for it even though they don't require such facility and performance of sub-system might go down.

2) **Ends are responsible for detecting failures/ crash:**
   I have developed ends in such a way that they are able to detect crash of each other.

3) **Trust:**
   I am keeping secret key at ends points which can be used for encryption/decryption of messages exchange between two entities.

4) **Performance:**
   Since I have opted for asynchronous interaction model, each process object is running on different system which would increase performance and no process gets blocked because of other process object.

5) **Scalability:**
   System is developed in a such manner that any number of process objects can connect to master object and perform event ordering.

Encryption / Decryption

I have implemented simple encryption/decryption using ceaser cipher technique by adding/substracting some KEY value.

Details are as below:

Encryption: Clock value + KEY

Decryption: Clock Value – KEY