Scalable and Fault Tolerant Architecture for Event Processing Systems

Project Proposal

Supervised By: **Mr. Manusha Wijekoon (Ext.)**
Mr. K. Sarveswaran (Int.)

Group Members:

H.E. Martin	**060295J**
D.S. Metihakwala	**060305T**
D.M.R.R. Sampath	**060428X**
H.C. Randika	**060398D**

Introduction

Complex Event Processing (CEP) is one of the emerging fields in data processing recently. This is essential for Event driven systems with High input data rates and near zero latency (nearly Real-Time). The basic functionality of complex event processing is to match queries (called triggers) with Events and trigger a response (e.g.: Alert Message) immediately. These queries describe the details about the event that the system needs to search within the input data stream and these queries can be even complex collection of events. Unlike traditional systems which are operating with hundreds of queries running for short durations against static data stored in the system (E.g.: Typical Relational database systems), event driven systems operate with stored queries running constantly against extremely dynamic data. Actually an event processing system is the upside down version of a search system where data stored in the system are matched against incoming queries. Hence, CEP is used in systems such as Data centre monitoring, financial services, Oil & Gas utilities, Web analysis, etc. where extremely dynamic data is being used. Abstractly, the tasks of CEP are to identify meaningful patterns, relationships and data abstractions among unrelated events and fire an immediate response such as an Alert message.
Examples:

- Search a Document for a specific key word
- Radio Frequency Identification
- Financial market transaction pattern analysis

As event processing becomes more important in business applications, lots of university research projects are being focused on event processing. Software development companies also have identified the growing potential of event processing system development and they are also striding towards developing new event processing systems and frameworks. Therefore, though event processing is relatively a new area, already there are a few event processing systems developed by university research groups and few other companies as well. As an example “Esper” system and “NEsper” system by EsperTech Inc, which are Open source products and the Microsoft’s CEP framework, a commercial application which is still in the development phase.

Usually event processing systems require higher processing power, memory and IO. Hence these systems require Mainframes or supercomputers to operate efficiently. But these days’ enterprises are not expecting hardware requirements such as Mainframes or supercomputers for their applications. Instead their requirement is to use applications which can run on commonly available servers and workstations. To meet operational requirements while satisfying enterprise requirements, event processing systems need to

have a relatively cheaper way of generating a higher processing power, memory and IO. Distributed systems are one of the best possible solutions for this problem. Hence almost all the event processing systems are designed focusing on distributed architectures. Above mentioned Esper and NEsper systems use JBoss, IBM Web sphere, GlassFish, Oracle Weblogic type of application servers as their containers and use application server clustering as their clustering mechanism to support distributed architectures. Our project is mainly aimed to create a scalable and fault tolerant distributed architecture which is specifically designed for complex event processing systems.

Problem Definition

A Distributed system is a system that uses multiple hardware units, typically hosts that are used collectively to provide a service. These hardware units can be hosts, networks, routers, switches, etc. In a typical enterprise level distributed system, these hardware units are connected using a high speed network for intercommunication. Functionality of the system is distributed among many processes. These processes can run on many machines, depending on their requirements on processing power, memory and IO. Distributed systems are a form of parallel processing. Multicomputer Distributed systems allow us to create redundant processing units with same computer programs running in different hosts. So each machine can be used to process same input with any difficulty. In this scenario, distributed systems can work as a mirror processing unit for the original processing unit. Hence such distributed system can be used for load balancing. This complete distributed system can have a high throughput and low latency in processing an input queue.

Scalability and fault-tolerance are very important properties in mission critical software systems. Mission Critical software systems refer to the systems which are highly depending on the software to achieve complete systems success. Scalability is the ability of the system to handle increasing loads while maintaining other performance criteria such as throughput, latency and etc. Scalability is very important in enterprise scale applications as it allows smooth operation of the system throughout its lifetime, without requiring large upgrades when load is increased. Hence scalability is a very important factor in software development. Fault tolerance is another important quality, rather a property of a computer system. This means the ability of the system to withstand failures in its components and continue its tasks. A software system that provides its services without any loss of functionality or performance under a given type of fault is said to be fail safe for that fault. If the system is operating with reduced functionality and/or performance it is said to be fail soft for that fault. If the system could not provide its services, it is

said to be fail stop system for the fault in concern. Generally, architecting a failsafe/fail soft systems are much harder than fail stop systems. Normally software supports this fail- soft requirement up to a given number of failures. They can carry out their work in a lower rate until there are, this given number of faults occur in the system. The performance downgrade is usually proportional to the significance or severity of the failure.

In an event processing system, each query must be executed against all incoming events and this requires a lot of processing power in large-scale enterprise systems where millions of queries have to be executed against a large stream of events.

For most enterprise applications, occurrence of events are unpredictable and the system should be able to handle a growth of the number of incoming events, since usually those systems are mission critical , real time systems, and should also be able to support a large number of queries accordingly, which means that the system must be scalable. Usually the enterprise systems are expected to be almost 100% uptime and thus the system must be fault-tolerant.

Most of the modern enterprises expect to deploy such systems on commonly available servers and workstations with limited processing capabilities rather than on mainframes or supercomputers and this in turn limits the number of queries that can be stored/ number of events that can be processed in one machine during a given time period. Therefore almost all EP systems are architected as distributed systems. However this solution introduces problems that are inherent to distributed systems such as unreliability in communication, unreliability of hosts etc.

Since event processing systems are usually mission critical and real-time systems, it is of great importance for such distributed system to be scalable and fault-tolerant. However, with the issues inherent to distribute systems, this is never an easy goal to achieve. Also, since the event processing market is still gaining popularity, there are not too many resources available for the event processing systems designers and developers at the moment on this area such as reference architectures which address those issues. Therefore it can be of great importance to introduce architecture for scalable and fault-tolerant distributed event processing systems which can address those issues.

Project Objective

Primary

Carrying out research and introducing an architecture for a scalable and fault-tolerant distributed event processing system, which can be used as reference architecture for building event processing systems.

Doing simulations using an appropriate toolkit or a language to demonstrate the characteristics of the proposed architecture.

Secondary

A direct small scale implementation of the proposed architecture that captures the scalability and fault-tolerance (optional).

Development of a fully featured framework for building distribute event processing systems (optional).

Methodology

For fulfilling the primary objectives, first we need to have a proper understanding about the specific requirements of event processing systems, architectures, characteristics of distributed systems and issues related to them. We also intend to study existing solutions given to this problem, their characteristics and drawbacks.

The new architecture can either be a modification to an existing one or it can also be a completely new one. The architecture shall be chosen depending on how well it fulfills the given requirements according to the research done in the first phase.

The next task of the first phase will be to simulate the proposed architecture to demonstrate its characteristics – efficiency, scalability, fault-tolerance etc. This will be done using a simulation platform such as Matlab/Simulink or using a common programming language, depending on their applicability for the task.

For the second phase that is for fulfilling secondary objectives we will be doing a direct implementation which captures the characteristics of the proposed architecture. We will use C#.NET or Java as the

programming language, and will make use of a suitable middle-ware platform such as CORBA or RMI if required.

For the third phase, which is an optional one, we will implement a fully featured framework taking the proposed architecture as the reference, which can be used for the development of event processing systems, either using .NET or Java.

Project Scope

As mentioned above this project aims to build a Scalable and fault tolerant architecture for event processing solutions. In general, events can be of many types, documents, multimedia content etc, but we will not assume any specific type during the project. Rather, the system will be architected to handle any event/query type in a generic manner. The actual pattern matching part is not considered in the project scope as well. Instead the primary goal of the project would be based on the task of designing and implementing an optimal scalable and fault tolerant distributed system architecture, thus enabling this architecture to be used in any existing or future complex event processing system.

The project would be divided in to three major phases. In the initial phase we intend to work on coming up with an architecture and simulating the proposed system to reflect its characteristics as a proof of concept. This project phase will enable us to identify and possibly correct any faults in the proposed architecture. Hence it will enable us to save a lot of time in not having to make changes to the architecture at later stages of implementation. Thus we would be able to proceed to the next phase with confidence in our solution.

The second phase of the project will consist of a small-scale direct implementation of the proposed architecture which would capture the scalability and fault tolerant nature of the system. It would consist of a small scale distributed system in order to demonstrate the actual workability and functionality of the architecture. The implementation would be done using a modern programming language and a middleware component which would be chosen depending on the language. This project phase would enable us to observe how the proposed architecture works on actual computers and how it behaves under different faults and different scales. This would give us valuable insight before moving towards the implementation of the actual framework.

The final phase of the project would be to implement a framework on which various event processing solutions can be developed on. The framework would be a fully functional solution reflecting the developed architecture which would act as a base for future event processing systems.

The Scalability of the system would be handled within two aspects, Events and Triggers. The system would be able to handle a theoretically unlimited number of events and triggers. The scale of the implementation would grow with the number of events and triggers but the architecture of the System would remain unchanged.

Deliverables

Initial Phase

1. Coming up with an optimal scalable and fault tolerant architecture for the system.
2. Implementing a simulation to reflect the characteristics of the architecture.
3. Testing the simulated environment in various scales with possible faults.
4. Making relevant modifications and finalizing the architecture.

Secondary Phase

1. Implementing a small scale distributed system which reflects the designed architecture.
2. Testing various scenarios on the implemented system.
3. Obtaining measurements of the actual performance of the architecture.

Final phase

1. Implementing a fully featured framework for the designed architecture.
2. Testing the architecture and correcting any faults to assure flawless functionality.

Challenges

Developing a distributed system is a tricky job. Achieving reliability is a main challenge of a distributed system design. But for a system to be truly reliable, there are other characteristics that need be addressed correctly. Following are the general challenges that we need to address in a distributed system.

- Fault tolerance: - Means that the overall system should have capability to recover from a failure that might happen in a single component or few components and continue the tasks without any performance shortage or a complete system failure.

- Scalability: - Mean that the systems' ability to handle increasing loads while maintaining the desired performance criteria such as throughput and latency. So we need to address typical scenarios like increase in size of the network which the system running, increase the number of users or servers, or overall load on the system.

- Recoverability: - This system should have the capability of recovering from failures like components can restart themselves and rejoin the system, after the cause of failure has been repaired.

- Highly availability: - The system should available for users when they need it. This implies that the system should resume providing services even when some components have failed.

- Consistency: - The system can coordinate actions by multiple components often in the presence of concurrency and failure.

- Predictable Performance: This implies that the system should have ability to provide desired responsiveness in a timely manner. Mainly consider about the throughput of the system.

- Security: Means that the system must authenticate the accesses to data and services.

Initially we learnt about two distributed system architectures. Following are description of them.

Architecture 1:

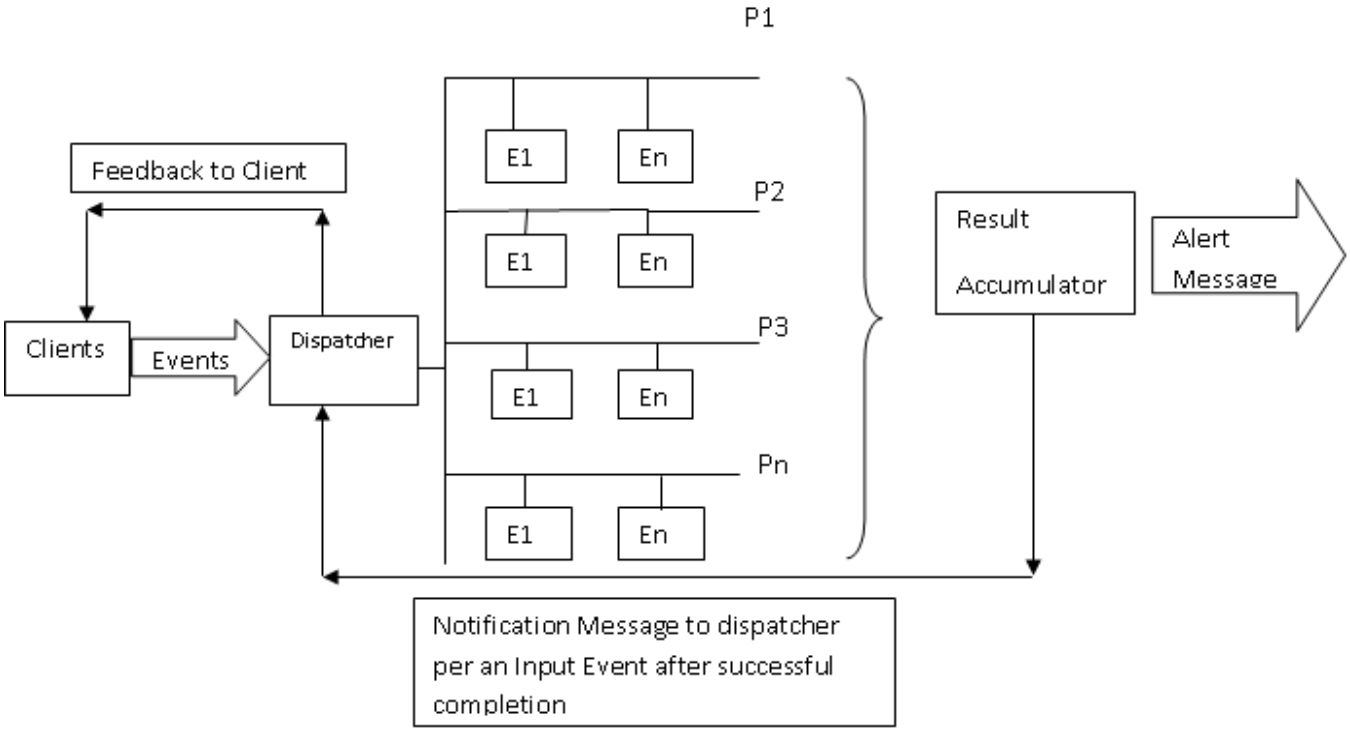


Figure : 1

Figure 1 shows the architecture of the first method we discussed. Dispatcher is used to distribute the queries (Trigger Base) to each partition. P1, P2, P3… Pn are the partitions where the trigger base is distributed by Dispatcher. E1, E2, E3… En denotes the engines which use to store same set of queries. Each Engine is identified by partition number and Engine number. As an example P1E1, P1E2….P1En are the addresses of the engines in the first partition. All the engines in a row contain the same queries. As an example let's say query 1 and 3 stored in 1st partition and query 2 and 4 stored in 2nd partition. When the number of queries increased, the queries are added to the each partition while considering the queries per partition. And if the number of partitions is less than required for new queries to store, a new partition with engines will be added to the system to store the queries (triggers). Each event comes to dispatcher sends to the one engine of each row and eventually it will cover the whole trigger base. Finally the output from each partition is taken and it is sent to the result accumulator. It will construct the output message (Typically an alert message) and will be sent to the desired destination. And meanwhile, a notification message for a successful alert message will be sent to the dispatcher by result accumulator and dispatcher will dispatch this feedback message to the event sender.

Problems:

We need to address component (engine) failures in the system. There should be a separate method to identify whether all the components are working properly. So we need to have message passing mechanism between the components (heart beat/pulse). Because these things can be only identify through a time out situation.

Another thing is failure to send or receive messages primarily due to lack of buffering space, and which might causes a message to be discarded with no notification to either the sender or receiver. This can happen when load balancing problem. So we need to address load balancing of the system effectively.

Network partitions are sub-networks within which messages can be sent, but between which messages are lost. This is called network partition failure. So we need to address this problem also.

Timing failures and Network failures are other types of failures that might occur in a distributed system. So we need to address them also in a fail safe manner.

Architecture 2:

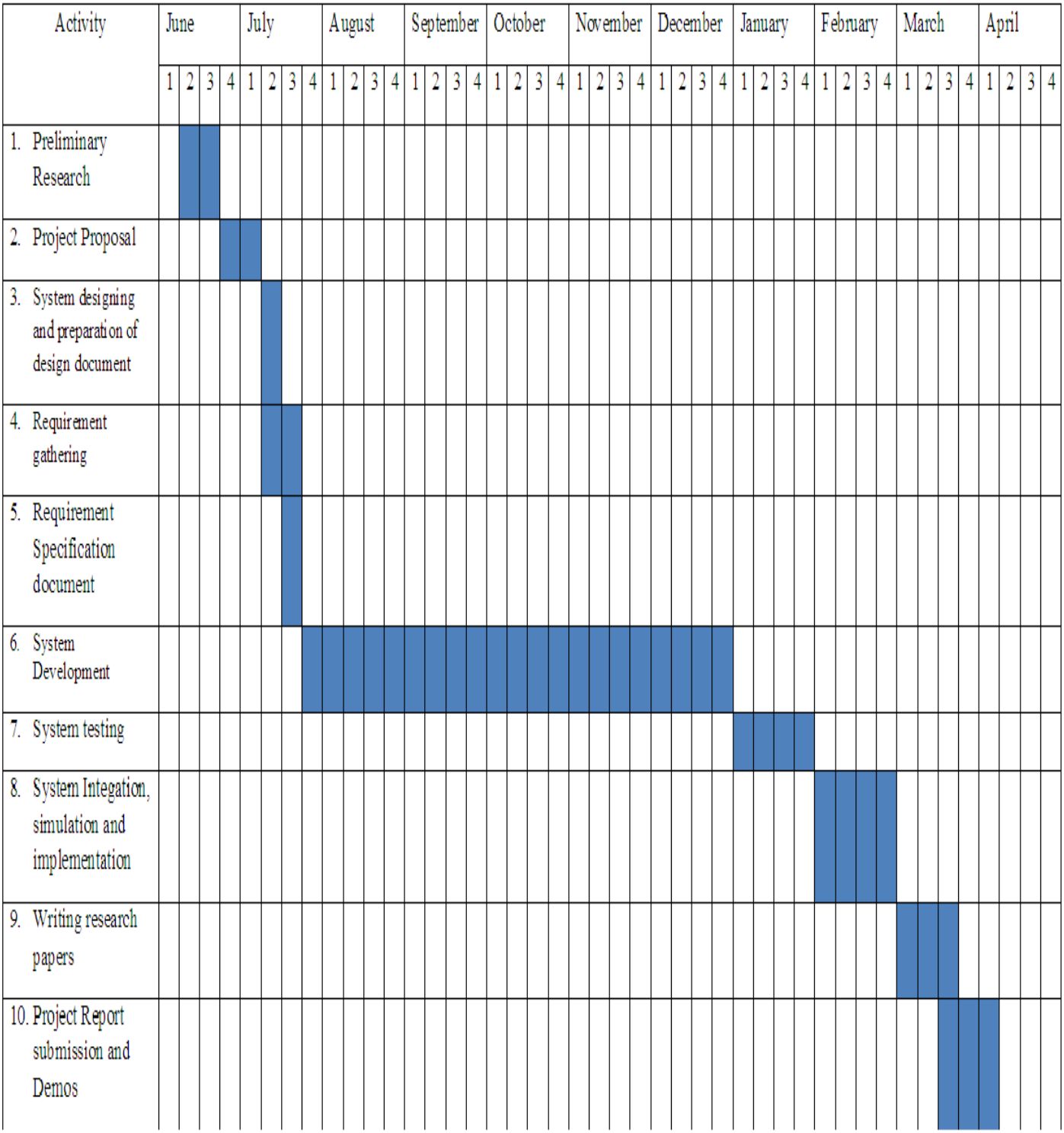
In this architecture there is a set of engines to store queries (triggers) given by the clients. In this scenario the big problem is trigger base become highly fragment in long run of the system.

Problems:

Challenge in this case is finding a solution to defragment or reduce the fragmented space in the trigger base.

So our challenge is to come with an architecture which is capable of overcome the problems exist in the above mentioned two architectures.

Project Plan and Schedule



Bibliography

Complex Event Proceesing-Wikipedia . (n.d.). Retrieved 07 01, 2009, from http://en.wikipedia.com: http://en.wikipedia.org/wiki/Complex_Event_Processing

Distributed systems -Wikipedia. (n.d.). Retrieved 07 02, 2009, from http://en.wikipedia.com: http://en.wikipedia.org/wiki/Distributed_system

Fail Safe - Wikipedia. (n.d.). Retrieved 07 04, 2009, from http://en.wikipedia.com: http://en.wikipedia.org/wiki/Fail_safe

Fault Tolerance System -Wikipedia. (n.d.). Retrieved 07 02, 2009, from http://en.wikipedia.com: http://en.wikipedia.org/wiki/Fault-tolerant_system

Scalability- Wikipedia. (n.d.). Retrieved 07 01, 2009, from http://en.wikipedia.com: http://en.wikipedia.org/wiki/Scalability