PAPER SUMMARY

The goal of *CX*, a network-based computational exchange is to provide a completely abstract and simple interface to any application programmer to design and deploy computationally intensive applications using the processing power available on computers on the internet. The application programmer must be free from handling inter-process communication and fault tolerance and performance issues that include heterogeneous operating systems/machines, communication latency, scalability and robustness.

The basic entities in the architecture of *CX* are

- *Consumer* : Entity seeking the computational resource
- *Producer* : Entity offering the computational power
- *Task Server* : Entity that coordinates the distribution of tasks from a set of consumers to a set of producers
- *Production Network* : A network of task servers and their producers which appear as a single entity to the consumers

Here the *Task Server* is the abstraction through which the consumers communicate with the producers. From the consumer's perspective it only sends a task to the task server and gets back the result when it becomes available.

In the isolated cluster model :

- The producer registers with the server and gets a proxy which downloads tasks from the server.
- The task is computed and removed from the server. If the producer encounters a error during execution the task gets reassigned to another producer until its successfully completion.
- To improve communication latency the task server proxy maintains a task cache where the results of the task execution are locally cached if they are subtasks. The proxy also maintains a ready task heap where the ordering of tasks is done depending on how many times the task has been assigned. A task assigned multiple times gets higher priority.
- When the number of tasks in the task cache falls below a particular threshold, the proxy prefetches tasks from the task server to prevent the producers from idling.

The paper also aims at handling fault tolerance of task server failures. When a task server fails the computation should proceed without recomputations. One way to do this is to organise the server network as a sibling fat tree. The siblings of a server in the tree form a mirror group. Every state change to a server is mirrored.

For efficient code distribution, each task server becomes a download location for task classes. The *CX* class loader downloads task classes to the primary root task server via the consumer's class loader. From here the classes are loaded down through the task server tree. Thus, *CX* therefore can be used in a variety of environments from a university to a corporate producer network.