Summary of

CX: A Scalable, Robust Network for Parallel Computing

CX is a network-based **C**omputational e**X**change which integrates a combination of variations of other researchers' ideas, such as work stealing of non-blocking tasks, eager task scheduling, and space-based coordination.

The goal of CX is to provide a simple and compact API that cleanly separates application logic from the logic that handles interprocess communication and fault tolerance. All of this should make it easier for the application programmer to design and deploy computationally intensive applications utilizing the processing power available on the computers on the internet.

The administrative complexity associated with multiple hardware platforms and operating systems is solved by using a virtual machine (Java VM) which provides a homogeneous platform on top of the heterogeneous sets of machines and operating systems.

The basic entities in CX' architecture are:

- **Consumer (C):** A process seeking computing resources
- **Producer (P):** A process offering computing resources.
- **Task Server (S):** A process that coordinates task distribution among its producers.
- **Producer Network (N):** A network of task servers and their associated producers.

The servers decouple communication so consumers and producers don't need to know each other or be active simultaneously. A producer network also acts as a single entity in negotiations with consumers.

The consumer sends a computational task to the task server, and receives a callback when the result is available. Producers repeatedly fetches *ready* tasks from the task server and compute them. Tasks are *ready* for execution only when all necessary arguments (if any) have been received. The task is not actually removed from the server until a completion signal is received by the server, so transactions are not necessary: A task is reassigned until some producer eventually completes it successfully. When a producer computes a task, it'll result in either the creation of new subtasks and/or compute the arguments that are needed by successor tasks.

Tasks should not be decomposed into too small subtasks with too low computational complexity, as this will make the communication latency more prevailing. CX is therefore not advisable for computations with short-latency feedback loops.

One single task server can only service a bounded number of producers before becoming a bottleneck. It's also a single point of failure. A network of task servers solves these problems by retaining the functionality of the isolated cluster, in addition to balance the task load among the servers via a diffusion process. Fault-tolerance is handled by organizing the server network as a *sibling-connected fat tree*, and replicating a server's tasks to its sibling, so that when a server fails, the sibling is able to restore the state to a replacement server.