BACKGROUND
Genome sequencing technology has witnessed tremendous progress both in terms of throughput as well as the cost per base pair. However, when it comes to genome assembly, there still exists a dilemma in the face of this technology. On one hand, we have a number of assembly programs that can deliver excellent results for small genomes, which can assemble minimal-genome organisms on a single node but cannot scale up. In the other hand, distributed and parallel genome assembly workflows capable of assembling genomes of the size of the human genome have been reported recently. Using partitioned de Bruijn graphs we reduce memory requirements and enable parallel and distributed genome assembly in the cluster. Experimental results using the Bruijn graph technology have also shown that our framework can assemble a human genome dataset (GSM132) in 16.5 hours using two nodes and 12-16 memory (10 GB). It is important to mention that this experiment was done in 2013, and since then, significant advances in computing power have been made.

METHODITY
• Both scalability and memory efficiency can be achieved with partitioned graphs.
• Avoid loading entire genome into memory at once.
• Each partition can be processed independently—more parallelism.
• One node can process a partition.
• Slides a smaller window over the entire transcript set of a subreading.
• A 4-mer is hashed on the basis of the alphabetically smallest substring.

MOTIVATION
• Next Generation Sequencers: • High coverage.
• High throughput.
• High throughput.
• High throughput.
• High throughput.

• Bruijn HLT 3 billion read per unit 15.1 million gigabytes.
• Concatenation assembly assembling next generation and more demanding by the assembly and memory processing.
• Perfect: Most of the assemblies are either scalable, or memory efficient but not both.

• Examples: (1) Mantis: Uses Bloom filters, memory-efficient but distributed. (2) AbhySS: Reads distributed but not scalable. (3) SWAP: Scales highly scalable, but consumes memory footprint.

• Our motivation: To build a distributed, scalable and memory-efficient assembler.

IMPLEMENTATION
• Built on top of ZeroMQ [2] (see references) — an unobtrusive framework for concurrent and communication.
• Provides a debugging framework driven by试剂的Actor Model [3].
• No global state — all actor communications through atomic message passing.
• No locks, synchronization, or explicit-synchronization-induced deadlocks.
• Provides an embeddable framework for low latency transmission of small packets.
• Allows for efficient data transmission and non-blocking data read.
• Also provides an efficient message passing semantic between actors (Superset, Pregel, GraphLab and others).

The following figure chooses the overview of the architecture used:

A single master process which is responsible for the fulfillment of task synchronization between nodes and load balancing.

Each node reads a slave process consists of a Slave node manager, which spawns and manages workers.

All processes have access to a distributed file system and communicate messages through each other via a high-speed network.

CONCLUSIONS & FUTURE WORK
We introduced a distributed assembly framework that utilizes our own partitioning scheme for de Bruijn graphs to achieve both scalability and memory efficiency. Experimental results show that our framework significantly reduces the memory footprint while creating fast assembly.

In future, we plan to enhance our framework with efficient compression and to create a full pipeline.

REFERENCES

Acknowledgments
This work supported in part by the National Science Foundation under grants MRI-1118061, MRI-1052018, LA Basin of U.S. under grants DE-SC0004959, DE-SC0004960, and NW Fleet Foundation.