Future Directions in Abacus:
Data-parallelism and Fault-tolerance

Directions

- Support data-parallel applications.
  - E.g., nearest neighbor search, edge detection, image recognition, and finding association rules.
- Increase dependability by tolerating node faults.
- Study the interaction between data-parallelism and fault-tolerance.
- Carry these out while maintaining central Abacus theme: easing storage system management via adaptation.

Data-parallelism

- Motivation: growth in the prevalence of data processing workloads running on server clusters.
- Goal: apply Abacus techniques to data-parallel workloads.
- Approach: decompose application into a parallelizable scanning stage and a centralized merging stage.
  - Scanners often run on storage nodes, other nodes are recruited when expensive.
  - A merger aggregates partial results from the scanners.
  - A merger runs either on the client or within the server cluster.
- Abacus adapts the number of nodes running scanners (degree of data-parallelism) and the location of the merger object.

Fault-tolerance

- Motivation: distributing computation makes an application more susceptible to failure.
- Goal: ensure that an application on Abacus is no more vulnerable to faults with no increase in programming complexity.
- Approach: leverage extant checkpoint/restore facility.
  - Each node checkpoints state to a "buddy" node.
  - After failure, Abacus restarts failed objects on free nodes.
  - Distributed checkpointing is easy because scanners are usually stateless.
  - Abacus dynamically reevaluates when to take checkpoints.
  - Frequent checkpoints lead to faster failure recovery.
  - However, checkpointing takes time.
  - The right balance depends on checkpointing cost and probability of failure, both which can change at runtime.

Users run data-parallel apps which access data on server clusters. Clients are active collaborators with the clusters, taking part of the computational burden. Distributed computation is protected by a checkpoint and recovery component.