Scalable and Highly Available Fault Resilient Programming Middleware for Exascale Computing

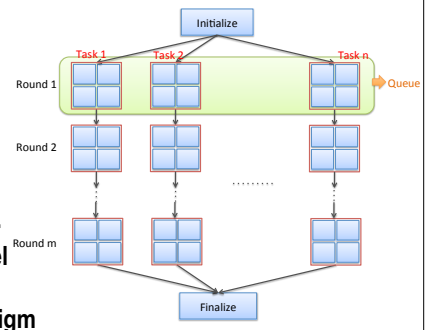
Abstract

Falanx is a programming middleware for the development of applications for exascale computing. Because of the fragility of the computing environment, exascale applications are required to be not only scalable, but also fault resilient. Falanx employs an MPI-based hierarchical parallel programming model for the scalability, where an application is described as a network of smaller tasks each of which are processed in a fine-grained parallel manner.
Falanx consists of Resource Management System (RMS) and Data Store (DS): The RMS allocates processes of each task to computing nodes avoiding nodes with failures. The DS preserves data required for each application, and prevents data loss due to failures. It is necessary that these components must be scalable and that they themselves have to be implemented in a fault resilient manner in exascale computing environments.
We design a *scalable and highly available middleware,* which consists of RMS and DS and implement them by using Apache ZooKeeper and Kyoto Cabinet. Then, we investigate the basic performance of RMS and DS from the preliminary experiments and confirm the feasibility of the middleware from an experiment using an actual application called OpenFMO.

Introduction

Background

- "Fault resiliency" is an important issue because MTBF of exascale-level computers will be short.
- Hierarchical master-worker (HMW) model is a promising programming paradigm that can achieve weak scaling on such computers.



An example of HMW programing model.

- We propose an MPI-based fault resilient programming middleware called Falanx.
  - Allows users to easily code a fault resilient HMW application.
  - Has been developing by using ULFM-MPI [Univ. of Tennessee, Knoxvile].

➡ Must be scalable and be implemented in a fault resilient manner as well.

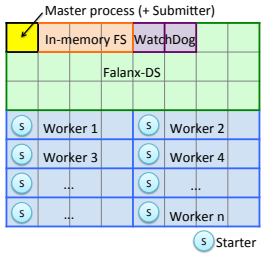
Contributions

- Design a *scalable and highly available middleware*, which consists of Resource Management System and Data Store.
- Implement RMS and DS by using Apache ZooKeeper and Kyoto Cabinet.
- Confirmed the feasibility from the preliminary experiments and an experiment using an actual application, OpenFMO.

Falanx: https://sites.google.com/site/spfalanx/

Overview of Falanx

- Falanx makes it easier to develop exascale applications, by providing a fault resilient runtime environment as a middleware.
- Focuses on a hierarchical master-worker (HMW) model.
- Falanx API allows users to
  - Code an application logic in the master process as workflow
  - Specify whether a failed task is restarted or destroyed when failure has happened.
- Falanx-based program is a single MPI job written in C.
  - High portability for supercomputing environments.
  - Each task is allocated to a computing node group, configured by an MPI communicator, respectively.
- Consists of Resource Management System (RMS) and Data Store (DS).



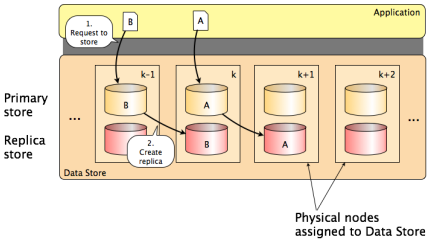
An example of MPI rank allocation

Data Store (DS)

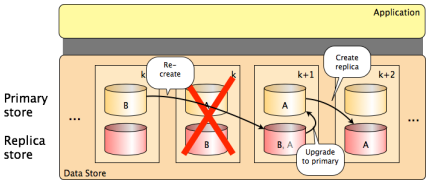
Functionalities

- User can store intermediate data of each task in DS.
- Provides high performance distributed on-memory storage.
- Automatically distributes data and their replicas over multiple DS nodes based on a simple hashing manner.
  - Data are managed redundantly for high availability.
  - Create a replica on a DS node next to the DS node, which stores its original data.
- Replicas are automatically reconstructed to prepare for multiple errors when a DS node fails.
- Initial access to the failed node is automatically redirected to the replica node.

DS creates a replica automatically.



DS automatically reconstructs a replica when failure happens.



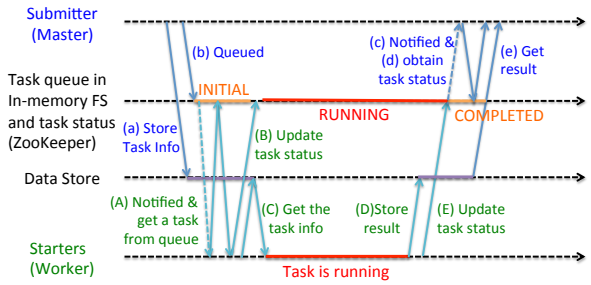
Implementation

- Falanx-DS is implemented by using Kyoto Cabinet.
  - Kyoto Cabinet: http://fallabs.com/kyotocabinet/.
  - GPL-based open source software.
  - Provides a C++-based light-weight DB library.
- Uses Kyoto Cabinet as an on-memory storage by disabling its persistence features.

Atsuko Takefusa, Tsutomu Ikegami, Hidemoto Nakada,
Ryousei Takano, Takayuki Tozawa and Yoshio Tanaka
National Institute of Advanced Industrial Science and Technology (AIST), Japan

Falanx API and Execution Sequence

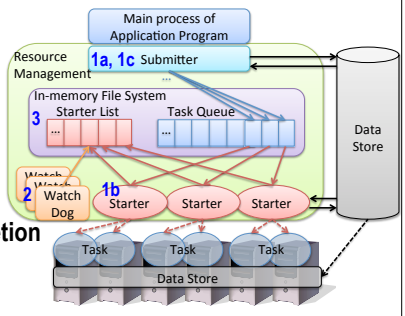
	Function Name	Description
(a)	data_store_set()	Store task information in DS
(b)	falanx_task_submit()	Submit a task to the task queue (Task status is set to INITIAL)
(c)	falanx_task_wait_any()	Notify of a task termination
(d)	falanx_get_result_value()	Acquire status of the terminated task (COMPLETED or FAILED)
(e)	data_store_get_value()	Acquire the task result from DS
(A), (B)	falanx_get_request_key_value()	Acquire the DS key of the allocated task and update status of the task (INITIAL to RUNNING)
(C)	data_store_get_value()	Acquire the task information from DS
(D)	data_store_set()	Store the task result in DS
(E)	falanx_set_response_key()	Store the DS key of the task and update the task status (RUNNING to COMPLETED)
	(WatchDog changes a failed task status to FAILED)	



Resource Management System (RMS)

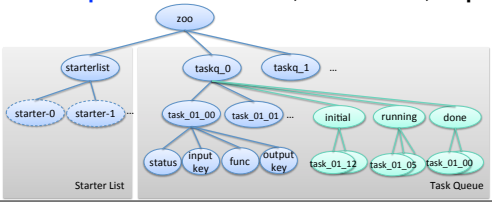
Functionalities

1. Task management by multiple processes.
   a. Task submission to a task queue.
   b. Execution of a task in the task queue.
   c. Re-execution or deletion of a failed task.
2. Health monitoring of computing nodes and the networks between them.
3. Persistent and scalable management of resource management information.



Implementation

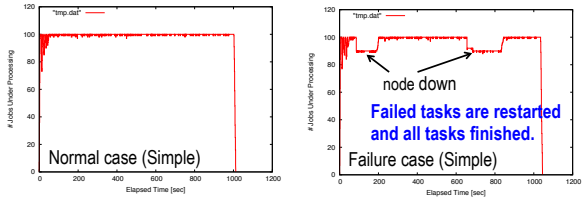
- RMS has been Implemented by using ***Apache ZooKeeper.***
  - Provides an in-memory file system over multiple nodes and an "watch" mechanism.
  - Multi op API can reduce communication overheads.
  - Does NOT provide a queue.
- Task queue is represented in ZooKeeper FS.
  - Naive: Each Starter autonomically accesses the queue.
  - Starter Queue: Less contentions, FIFO-based, w/o priority
  - Leadership: Less contentions, FIFO-based, w/ priority



Preliminary Experiments

I. Fault resiliency: Normal and Failure cases

(Settings: 100 Starters, 10sec(sleep)x10000 tasks)



II. Scalability of ZooKeeper in a HPC scenario

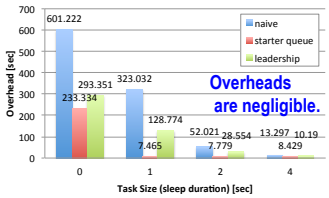
(Settings: 100 Starters, 10sec(sleep)x10000 tasks)

# of ZK nodes	3	5	7
Elapsed time [sec]	1023	1021	1021

Elapsed times are comparable.
→ ZooKeeper is scalable.

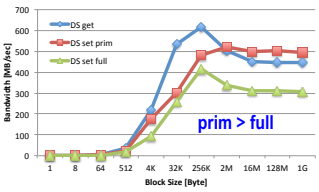
III. Difference between zk queue

(Settings: 60 Starters, 10000 tasks)



IV. Falanx-DS set/get BW

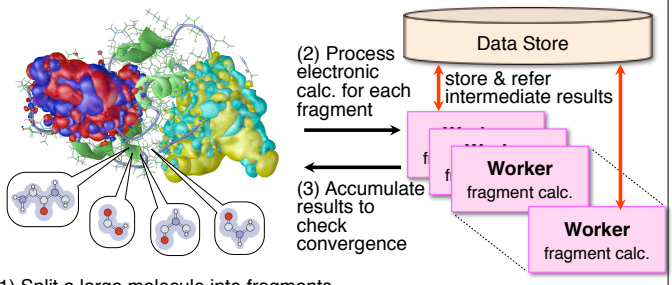
(wait for replication to complete: prim: NO, full: YES. InfiniBand 40G)



Experiments using OpenFMO

OpenFMO Application

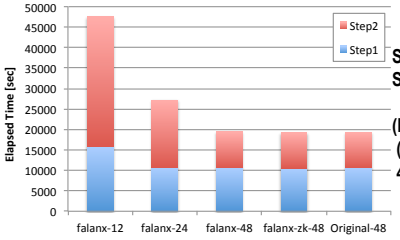
- OpenFMO is an open-source software platform for Fragment Molecular Orbital (FMO) method.
- FMO is focused on ab initio electronic state calculations of macro molecules.



(1) Split a large molecule into fragments

Experiments

- Strong scaling measurement on a lysozyme molecule w/o water
- Compare Original vs. Falanx (1 rank queue) vs. Falanx zk
- Total # of cores varies from 12 / 24 / 48 cores.



Step 1: 57 tasks x 21 iterations
Step 2: 365 tasks

(Intel Xeon CPU E5540 2.53GHz (4cores) x 2CPU, 48GB memory, InfiniBand 40G)

- The performance is suffered from a severe load-imbalance in Step 1.
- The results of Step 2 show a good scalability.
- We will implement a facility to rearrange the worker configuration.

This work was partly funded by KAKENHI 25871199.