HTCaaS: Leveraging Distributed Supercomputing Infrastructures for Large-Scale Scientific Computing

Jik-Soo Kim, Ph.D

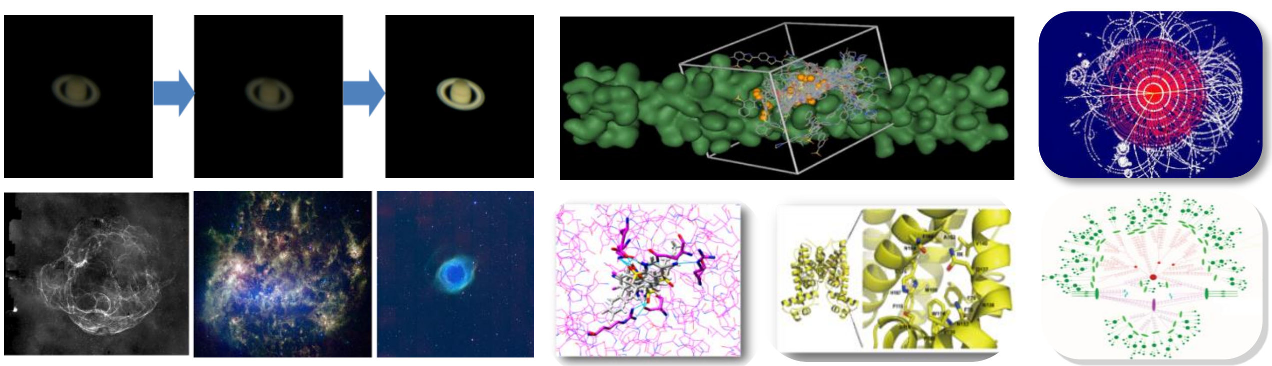
National Institute of Supercomputing and Networking(NISN) at KISTI

Table of Contents

- Introduction
- HTCaaS: High-Throughput Computing as a Service
- Evaluation
- Conclusions & Discussions

Introduction

- **From HTC to Many-Task Computing (MTC) [MTAGS'08]**
  - A very **large** number of tasks (millions or even billions)
  - Relatively **short** per task execution times (sec to min)
  - **Data** intensive tasks (i.e., tens of MB of I/O per second)
  - A large **variance** of task execution times (i.e., ranging from hundreds of milliseconds to hours)
  - Communication-intensive, however, not based on message passing interface (such as MPI) but through **files**



astronomy, physics, pharmaceuticals, chemistry, etc.

Introduction

➲ Middleware Systems for HTC/MTC applications

- ➢ **Ease of Use**
  - ✓ Minimize user overhead for handling jobs and resources
- ➢ **Efficient Task Dispatching**
  - ✓ The overhead of task dispatching should be low enough
- ➢ **Adaptiveness**
  - ✓ adjust acquired resources according to changing load
- ➢ **Fairness**
  - ✓ ensure fairness among multiple users submitting various numbers of tasks
- ➢ **Reliability**
  - ✓ Failed or suspended tasks should be automatically resubmitted and managed
- ➢ **Resource Integration**
  - ✓ effectively integrate as many computing resources as possible

Introduction

➲ Our Approach

➢ High-Throughput Computing As a Service

- ✓ *Meta-Job* based automatic job split & submission
  - e.g., parameter sweeps or N-body calculations
- ✓ Agent-based *multi-level scheduling*
- ✓ User-level Scheduling and *Dynamic Fairness*
- ✓ *Pluggable interface* to heterogeneous computing resources
- ✓ Supporting many *client interfaces*
  - Native WS-interface, Java API
  - Easy-to-use client tools (CLI/GUI/Web portal)

➢ HTCaaS is currently running as a pilot service on top of PLSI

- ✓ supporting a number of scientific applications from **pharmaceutical domain** and **high-energy physics**

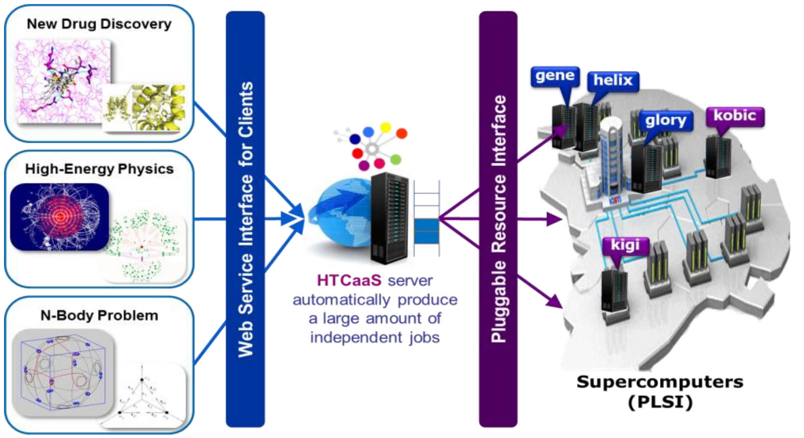
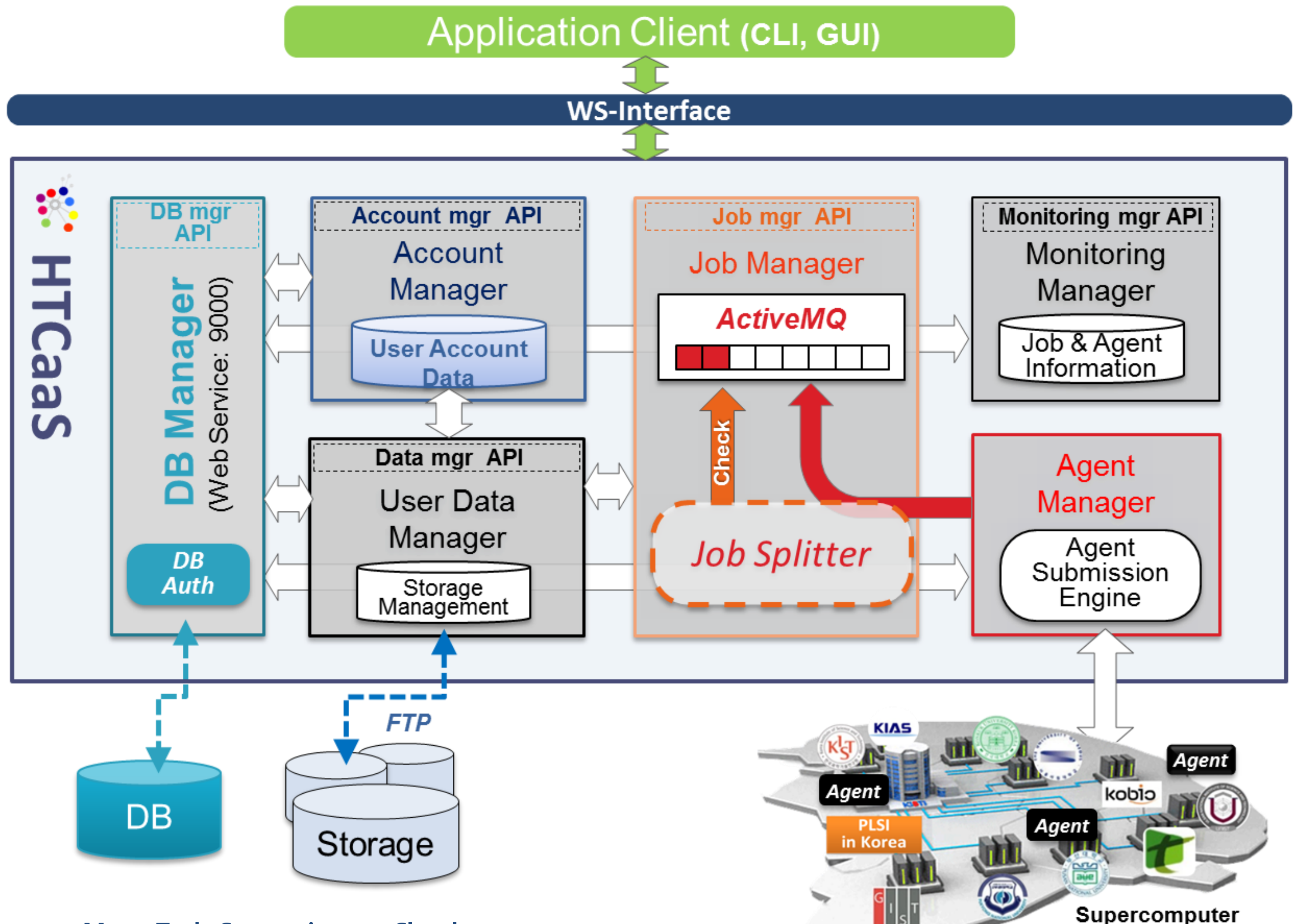


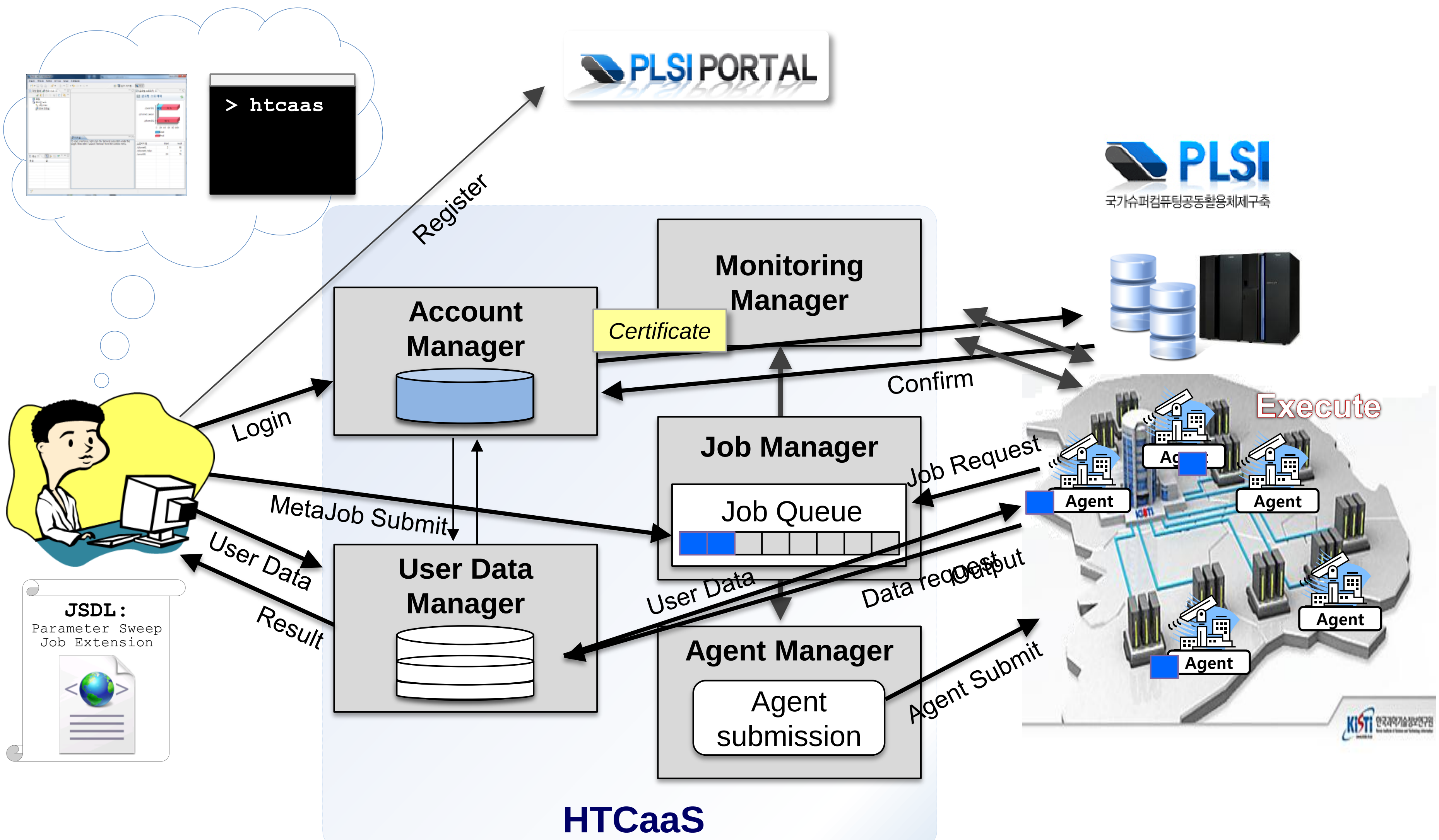
Table of Contents

High-Throughput Computing as a Service

➲ System Architecture



High-Throughput Computing as a Service



High-Throughput Computing as a Service

➲ Job Queues and Agents Management

- maintains **separate** job queues and agents **per** user
  - ✓ reducing complexities of *accounting* and *scheduling*
    - track and meter the usage of PLSI computing resources *per* user
    - carefully calibrating the number of agents per user can address the problem of *fair resource sharing* among multiple users
  - ✓ Each agent actively pulls the tasks from its dedicated job queue which corresponds to a specific **user**
    - if there are no more tasks to be processed, it automatically *releases* the acquired resources and exits

➲ Monitoring and Fault-tolerance

- periodically checks the status of agents and tasks
  - ✓ If some of agents or tasks fail, the Monitoring Manager informs the Agent Manager (or the Job Manager) to resubmit the failed agents (tasks) and manage them (addressing **Reliability**)

High-Throughput Computing as a Service

➲ User-level Scheduling and Dynamic Fairness

➢ Dynamic fair resource sharing algorithm

- ✓ divides all available computing resources *fairly* across all *demanding* users in the system (when the system is heavily loaded) and exploits *dynamic adjustment* of acquired resources as free computing resources become available (as the overall system becomes lightly loaded)
- ✓ Resource Allotment Function

$$RA(U) = min\left(NumTasks(U), \frac{AvailableCores}{\sum_{p \in DU} Weight(p)} * Weight(U)\right)$$

  - Weight(p) represents the *weight* of a user p
    - » can consider many different factors such as the number of tasks submitted by the user p, task running time, priority, etc.
    - » Weighted Fairness
- ✓ addressing **Fairness** and **Adaptiveness**

High-Throughput Computing as a Service

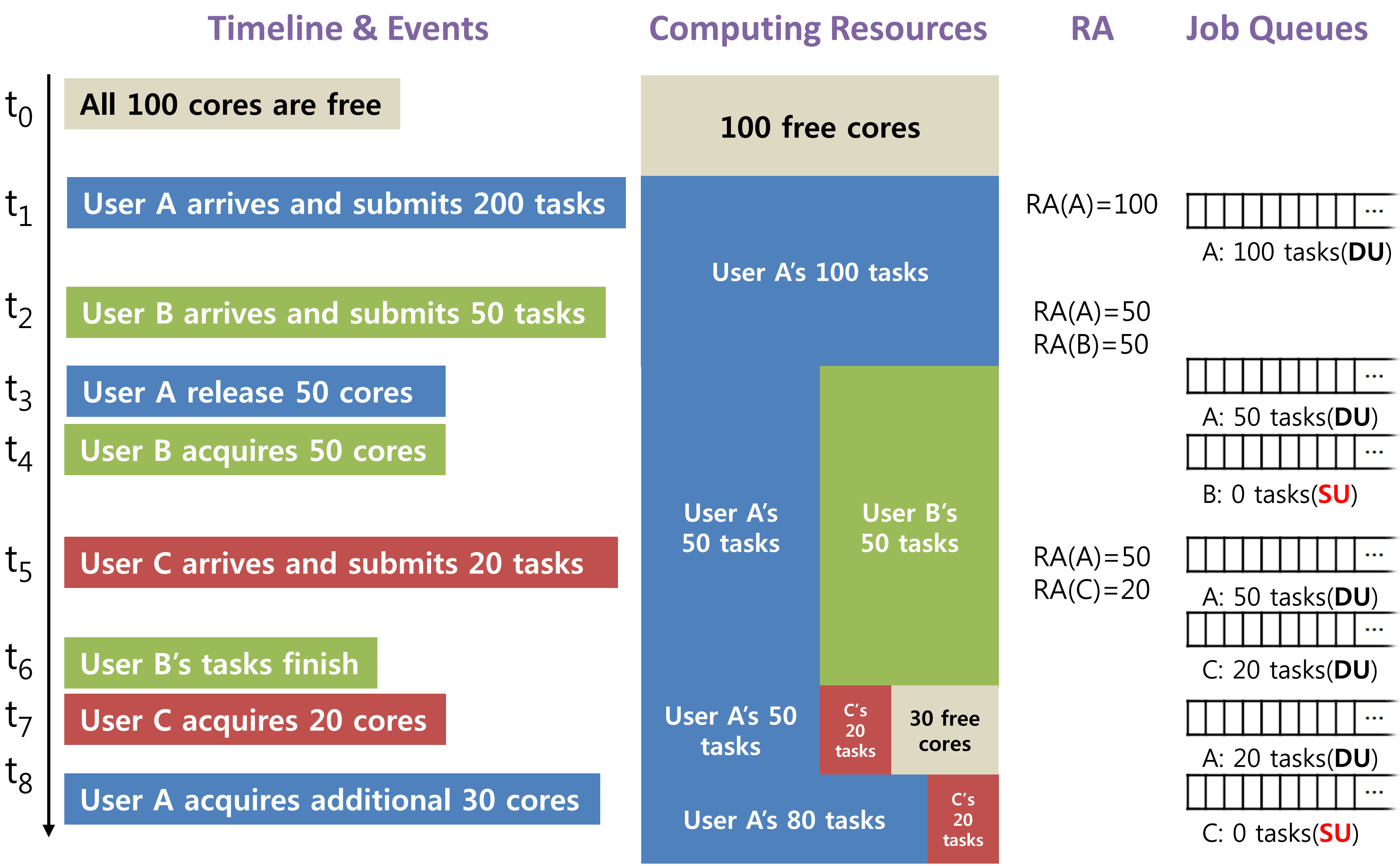


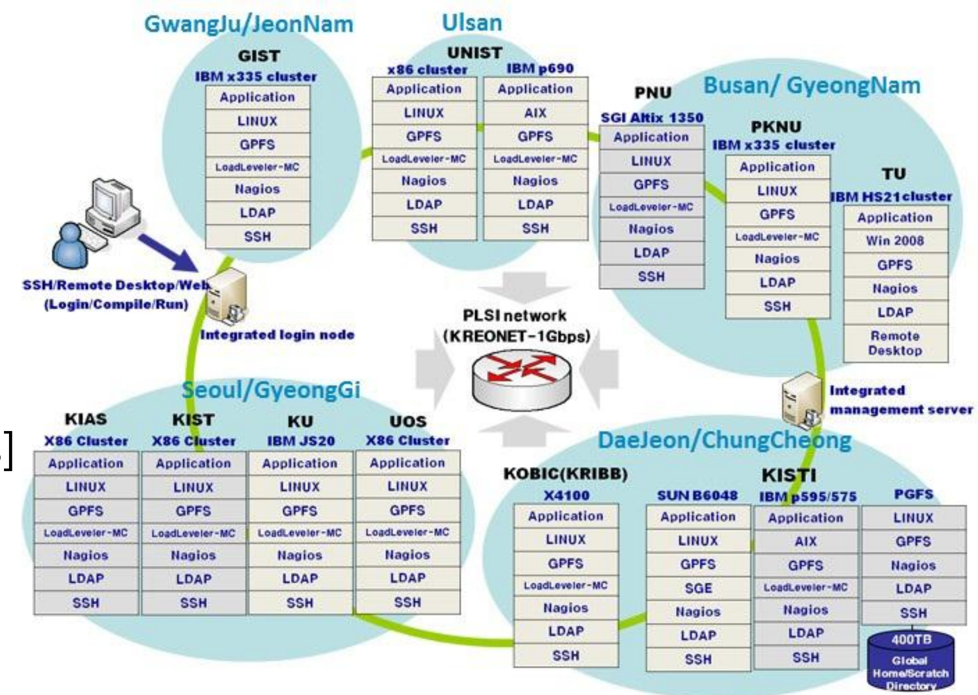
Table of Contents

Evaluation

➲ **PLSI (Partnership & Leadership for the nationwide Supercomputing Infrastructure)**

- provide researchers with an integrated view of **geographically distributed supercomputing infrastructures** to solve complex and demanding scientific problems

consisting of **multiple organizations** connected via a dedicated **1Gbps** network [**100TFLops** of computing power, 1,115 nodes with 8,508 CPU cores]



Evaluation

➲ PLSI provides a common software stack

- Accounting (based on LDAP), Monitoring (via Nagios), Global scheduling (based on **LoadLeveler**) and a Global shared storage system (based on **GPFS**)
  - ✓ utilize the LoadLeveler as a job submission system to available computing nodes in the PLSI
    - configured as a *multi-cluster* environment
  - ✓ exploits a total 400TB of global home/scratch directories mounted at *every* computing node as a shared storage system for input/output data and executables

ORG	*SYSTEM*	*PROCESSOR*	*NETWORK*	*OS*	*CORES*	*MEM(GB)*	*GFLOPS*
KIAS	helix (x86 cluster)	AMD Opteron 2GHz	1GbE	CentOS 6.2	128	8	1,024
KIAS	gene (x86 cluster)	AMD Opteron 2GHz	1GbE	CentOS 6.2	128	8	1,024
KOBIC	kobic (SUN X4100)	AMD Opteron 2.1GHz	1GbE	CentOS 5.4	184	4	1,545.6
KISTI	glory (SUN x2100)	AMD Opteron 1.8GHz	1GbE	CentOS 5.4	514	2	3,700.8

Table 1: PLSI Computing Resources leveraged by HTCaaS

Evaluation

➲ Micro-Benchmark Experiments

- simulating a large number of short-running tasks (sleep 10) and relatively long-running tasks (sleep 100)
- glory cluster in KISTI (300 cores)
- Performance Metrics
  - ✓ Makespan (time to complete a bag of tasks)
  - ✓ Efficiency (comparison with ideal parallelism)

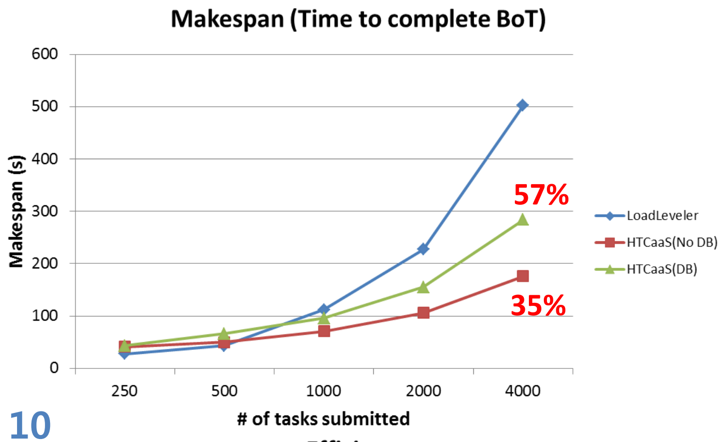
$$Efficiency(NT) = \frac{\frac{NT * PTE}{NCU}}{Makespan(NT)} * 100$$

  - ✓ Comparison Models
    - HTCaaS with DB Manager connections (**HTCaaS(DB)**)
    - HTCaaS without DB Manager interactions (**HTCaaS(No DB)**)
    - LoadLeveler+GPFS (**LoadLeveler**)

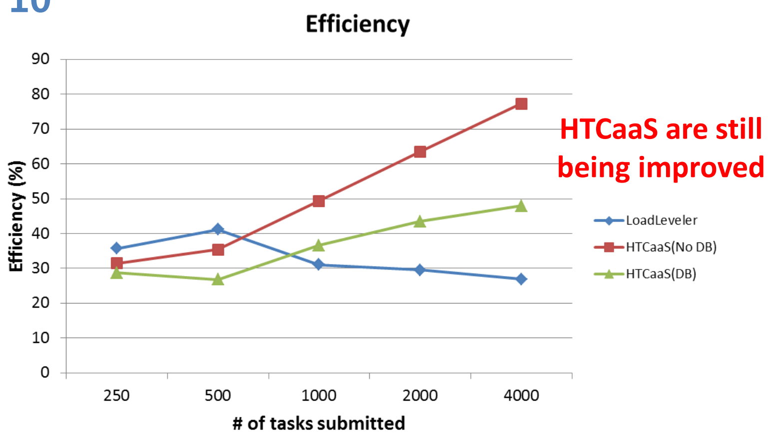
Evaluation

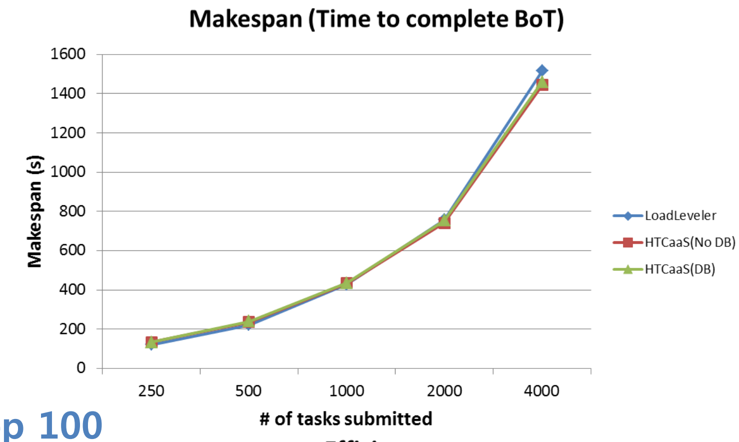
➲ Micro-Benchmark Experiments

- For short running tasks, HTCaaS clearly outperforms LoadLeveler
- For relatively long running tasks, overheads of task dispatching can be effectively countervailed

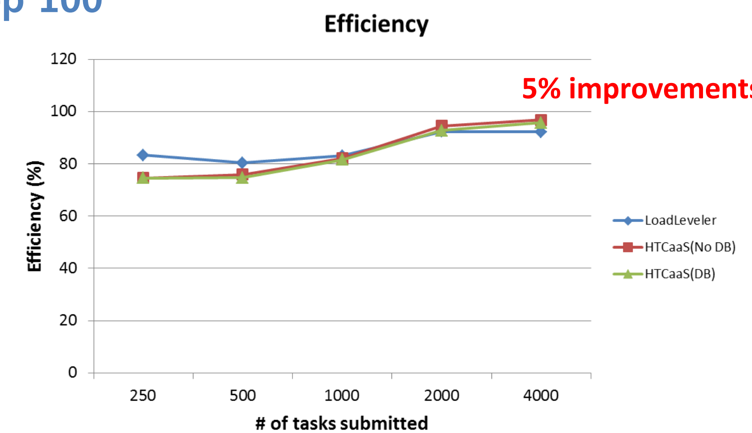


sleep 10





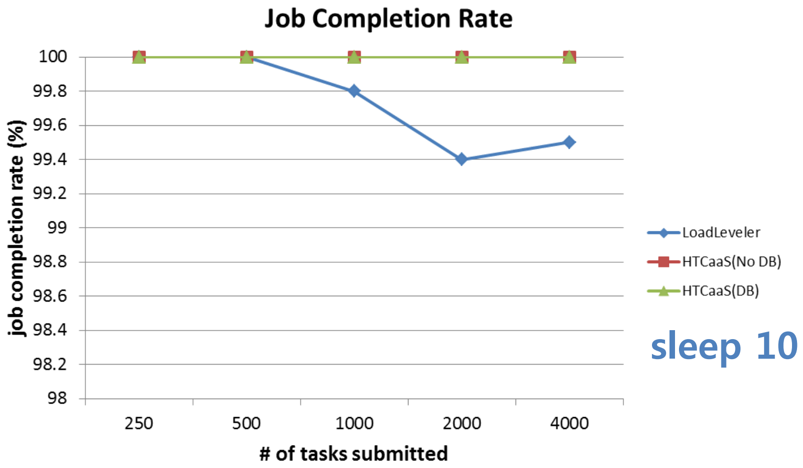
sleep 100

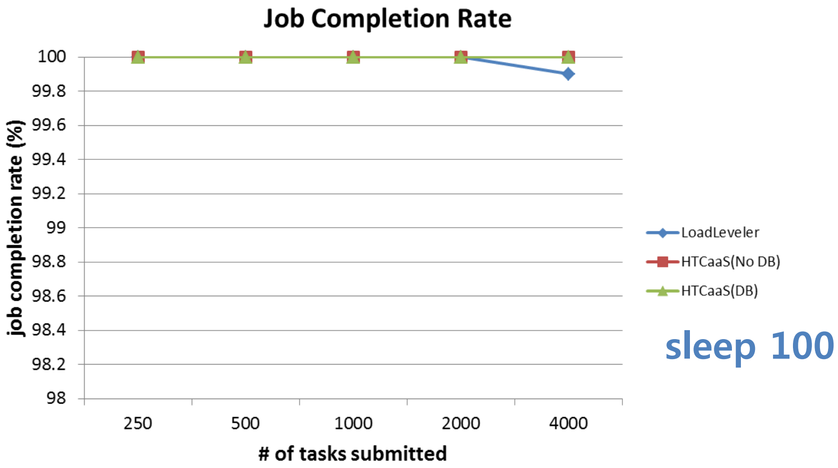


Evaluation

➲ Micro-Benchmark Experiments

- **Job holding problem** during the course of LoadLeveler dispatching
  - ✓ due to multiple simultaneous I/O operations on the GPFS
- HTCaaS can effectively leverage the **local storage** of the computing resource
  - ✓ User Data Manager manages overall input/output data staging
    - support data-intensive HTC/MTC applications where typical size of a single input data is relatively small (from hundreds of KBs to MBs)





Evaluation

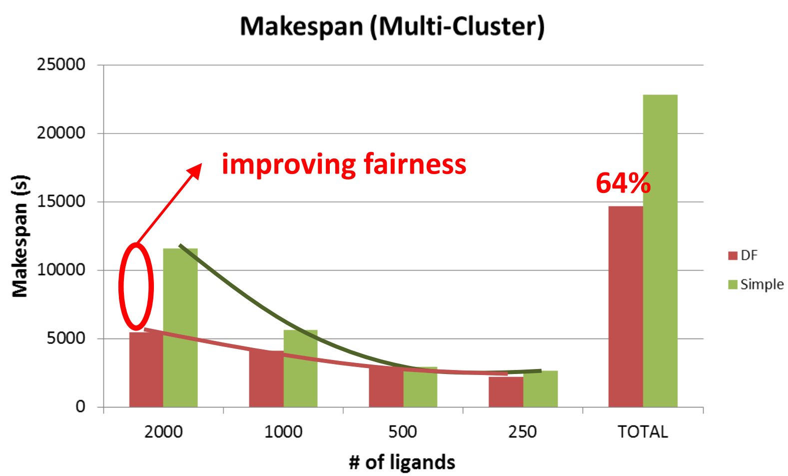
➲ Protein Docking Experiments

- Autodock, a suite of automated docking tools
  - ✓ perform the docking of *ligands* to a set of target *proteins* to **discover new drugs** for several serious diseases such as SARS or Malaria
- Experimental setup
  - ✓ Clusters: glory, kobic, gene and helix
  - ✓ **Four** different users are sequentially arriving at our system and submit various numbers of tasks (ligands) with an average inter-arrival time of 10 minutes
    - from 1000 down to 100 for the single cluster
    - from 2000 down to 250 for the multi-cluster
  - ✓ Comparison Models
    - HTCaaS with dynamic fair resource sharing algorithm – *dynamic* fairness (**DF**)
    - Simple resource partitioning algorithm – *strict* fairness (**Simple**)

Evaluation

➲ Protein Docking Experiments

- HTCaaS can *reduce* the performance gap among users with various numbers of tasks
- HTCaaS can *seamlessly* integrate multiple geographically distributed clusters
  - ✓ fully utilize available computing resources as they become available, while Simple inevitably wastes idle computing resources



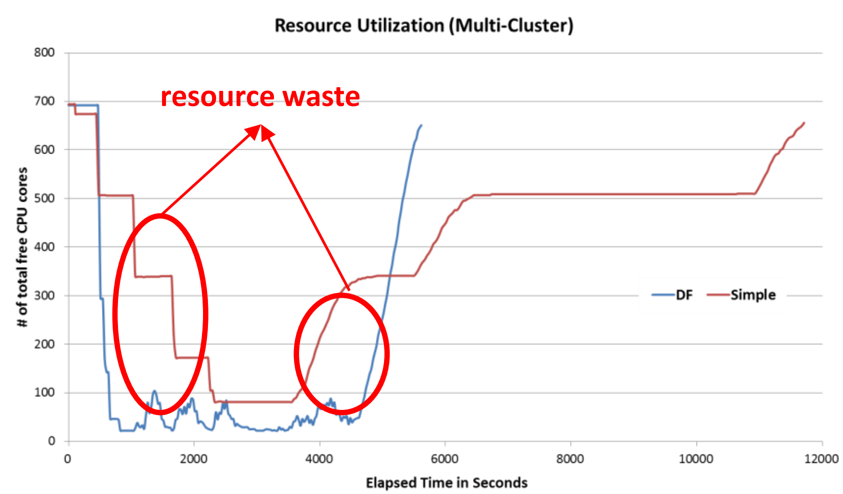


Table of Contents

Conclusion

➲ **HTCaaS to provide researchers with ease of exploring large-scale and complex HTC/MTC problems by integrating distributed national supercomputers**

- Employing the concept of **Meta-Job** (with easy-to-use client tools) and a **fault-tolerant agent-based multi-level scheduling** mechanism (**Ease of Use**, **Efficient Task Dispatching** and **Reliability**)
- Employing **dynamic fair resource sharing** mechanism (**Fairness** and **Adaptiveness**)
- HTCaaS effectively **leverages local disks** of geographically distributed computing resources
  - ✓ support data-intensive HTC/MTC applications
- HTCaaS can **seamlessly utilize** all of available computing resources without resource wastage (**Resource Integration**)

Discussion

➲ Future Work

- supporting more complex workloads consisting of HTC and HPC tasks
- improving the scalability of HTCaaS, and applying job profiling technique to realize the weighted form of fairness
- How we can more efficiently support *data-intensive* MTC applications on top of *geographically distributed* computing environments such as PLSI?
  - ✓ Shared storage system such as GPFS will be performance bottleneck
  - ✓ Data caching may work but it depends on the workload characteristics


Thank you!
National Institute of
Supercomputing and Networking
2013