TEXAS TECH UNIVERSITY

# CS4379: Parallel and Concurrent Programming
# CS5379: Parallel Processing

## Lecture 13

**Dr. Yong Chen**

**Associate Professor**

**Computer Science Department**

**Texas Tech University**

# Course Info

- **Lecture Time**: TR, 12:30-1:50
- **Lecture Location**: ECE 217
- **Sessions**: CS4379-001, CS4379-002, CS5379-001, CS5379-D01
- **Instructor:** Yong Chen, Ph.D., Associate Professor
- **Email:** yong.chen@ttu.edu
- **Phone:** 806-834-0284
- **Office**: Engineering Center 315
- **Office Hours**: 2-4 p.m. on Wed., or by appointment
- **TA:** Mr. Ghazanfar Ali, Ghazanfar.Ali@ttu.edu
- **TA Office hours:** Tue. and Fri., 2-3 p.m., or by appointment
- **TA Office:** EC 201 A
- **More info:**
  - http://www.myweb.ttu.edu/yonchen
  - http://discl.cs.ttu.edu; http://cac.ttu.edu/; http://nsfcac.org

# Outline

- Questions?

- Course research project - conducting research and development and becoming a lifelong learner

# Course Research Project

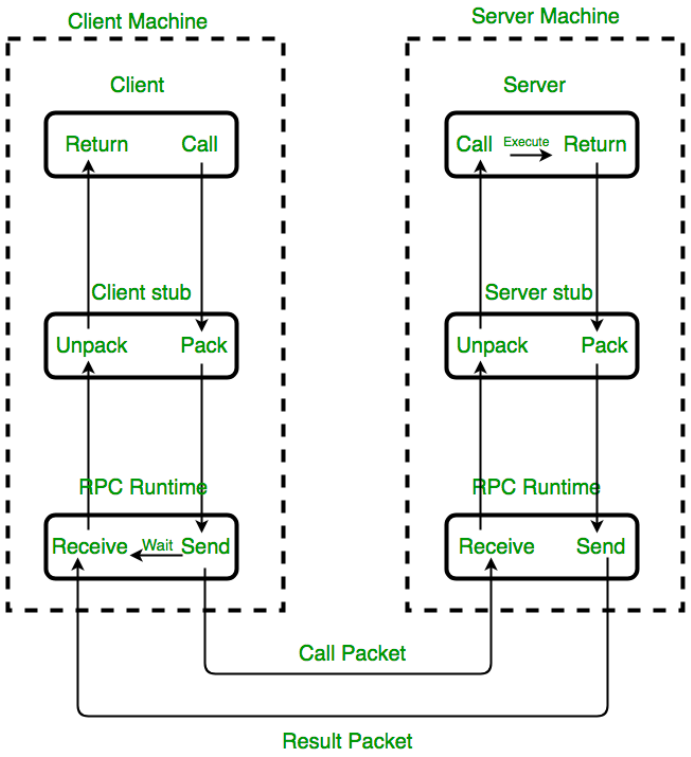
- Required for grad students, optional but encouraged for UG students (opportunity to work with mentors to learn beyond classroom and to publish papers and software)
- Topic selection and 1-page proposal due by 3/12
  - Need to approach the mentor and have his/her approval first, once you have the mentor's approval, topic selection is based on FCFS
  - https://doodle.com/poll/zvnsi395p52qdu63
- Final deliverables due: May 7th, 11:59 p.m.
- Please go through the documents on Blackboard in detail

# Topic: #1
# gRPC Data Streaming Performance

Dr. John Leidel
Chief Scientist, Tactical Computing Labs
Adjunct Researcher, Texas Tech
jleidel@tactcomplabs.com

# Traditional File System Implementations

- Distributed/parallel file systems utilize RPC in order to "trigger" remote operations
  - Data requests (read/write)
  - Attribute requests (metadata lookups)
  - Attribute updates (file time stamps, permissions)
- RPC system interfaces are difficult to implement
  - Require system-specific knowledge (Sys-V versus Linux verus Windows)
  - Implementations prone to errors, security holes
  - Performance often suffers




Implementation of RPC mechanism

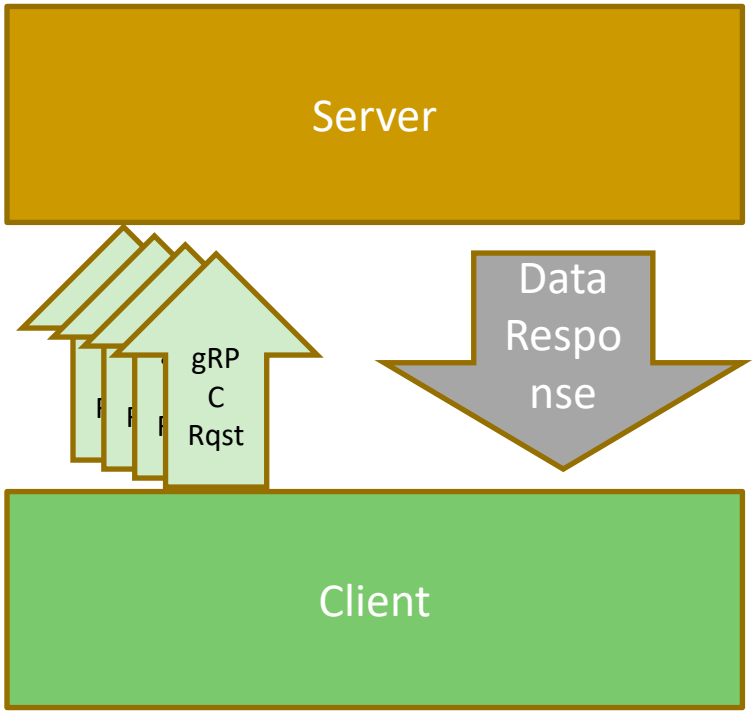
https://www.geeksforgeeks.org/remote-procedure-call-rpc-in-operating-system/

# Google RPC (gRPC)

- Google utilizes RPC extensively in their internal infrastructure services
- Google RPC (gRPC) was released open source in order to assist others to implement large-scale RPC-driven client-server applications
- Advanced features:
  - Native support for synchronous and asynchronous RPC calls (queuing)
  - Native support for advanced security and authentication mechanisms
  - High performance data streaming API
  - Very simple API mechanisms
  - Cross-language support (C++, Python, Rust, Go, etc)
  - Complete, cross-platform support (Linux, Unix, Windows, OSX)

# gRPC Data Streaming Performance Project

- Implement a client-server platform using gRPC using C++
- The client request data payloads from the server
  - Data requests of < 64 bytes - ~1GB (and ranges in between)
- The benchmark should record the latency and bandwidth of each transfer
- The client and server must execute on different physical machines
- Utilize the xBGAS-FS protocol as a sample protocol
- **Preferably implement both synchronous and asynchronous transfers
- **Goal: Determine if gRPC is suitable to implement a large-scale distributed, parallel file system**



# References

- gRPC [https://grpc.io]
- gRPC C++ Quickstart [https://grpc.io/docs/quickstart/cpp/]
- gRPC Full C++ API [https://grpc.github.io/grpc/cpp/]

- xBGAS File System (currently utilizes gRPC)
  - https://github.com/tactcomplabs/xbgasfs

# Topic: #2
# Atomic Memory Operation Benchmarks

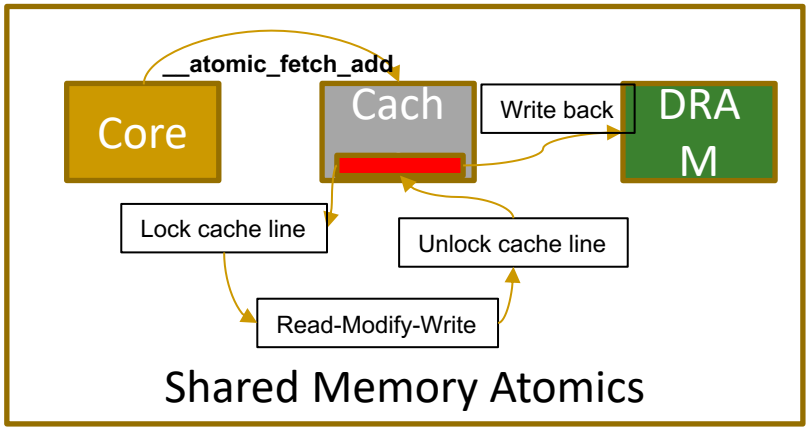
Dr. John Leidel
Chief Scientist, Tactical Computing Labs
Adjunct Researcher, Texas Tech
jleidel@tactcomplabs.com

# Concurrency Bottlenecks

- Large scale parallel applications often suffer from scaling issues due to the inability to efficiently implement concurrency
- This is commonly due to:
  - Wide use of barrier synchronizations
  - Serialization of computational operations (AllGather, etc)
  - Concerns over data atomicity (avoiding data race conditions and numerical consistency)
- Each of these components is often implemented using **atomic** operations
- Programmers are taught to avoid using atomics due to performance concerns…. **BUT WHY?**

# Atomic Operation Implementations

- Atomic operations are often implemented using a variety of cache-based techniques
- Shared memory atomics (eg, OpenMP, pthreads) are often implemented on cache lines
  - A single atomic memory operation “locks” a cache line, performs a read-modify-write and unlocks the cache line
- Distributed memory atomics are often implemented using combinations of barrier synchronizations and local atomics




*Oreste Villa, Gianluca Palermo, and Cristina Silvano. 2008. Efficiency and scalability of barrier synchronization on NoC based many-core architectures. In Proceedings of the 2008 international conference on Compilers, architectures and synthesis for embedded systems (CASES '08). Association for Computing Machinery, New York, NY, USA, 81–90. DOI:https://doi.org/10.1145/1450095.1450110*

# Atomic Memory Operation Benchmarks

- Utilize the CircusTent benchmark suite to determine the shared memory system performance of atomic memory operations
- CircusTent is written in C/C++ and contains 8 benchmark kernels implemented using two styles of atomics (AMO_ADD, AMO_CAS)
  - 16 total benchmarks
- Execute the OpenMP version of the benchmark for at least 5 DIFFERENT platforms
  - Laptops, embedded platforms, different servers, etc
- Execute the benchmark from 1 to N cores on the system (and all core counts in between) using 50% of the total memory capacity
  - We want the ability to graph the scaled performance
- Record the performance parameters for each platform
  - Timing: runtime in seconds
  - GAMS: “billions of atomic operations per second”; normalized metric
- **Optional: Port the benchmark to a new platform (eg, GPU)


https://github.com/tactcomplabs/circustent

# References


- CircusTent Benchmark [https://github.com/tactcomplabs/circustent]
- GNU Atomic Built-ins [https://gcc.gnu.org/onlinedocs/gcc/_005f_005fatomic-Builtins.html]

# Topic: #3
# Data Collection from Lustre File System I/O Operations

Mr. Misha Ahmadian
Research Assistant at HPCC
Ph.D. Student in CS

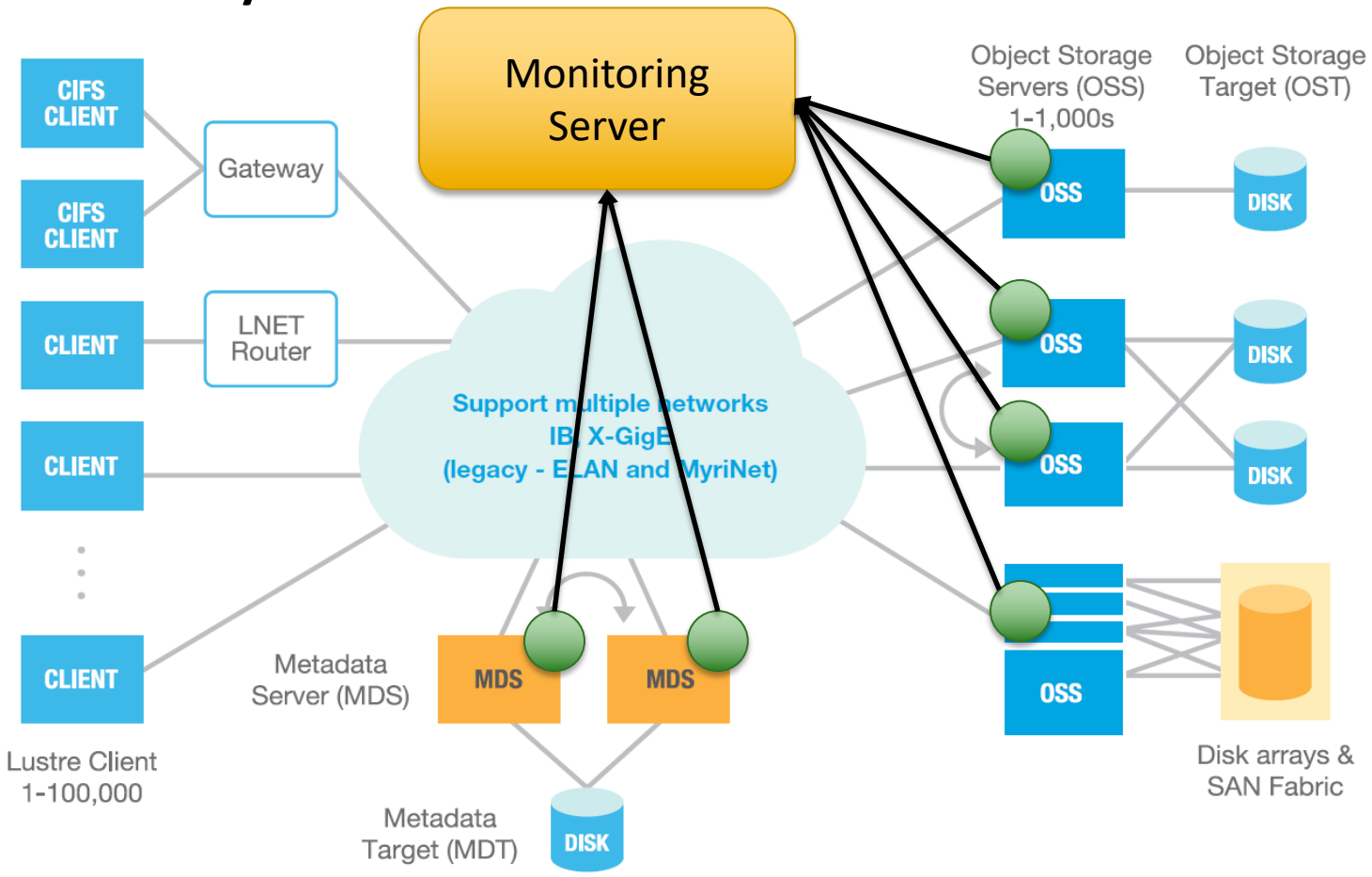
# Collecting I/O Statistics from Lustre Parallel File System

- **What is Parallel File System?**
  - Stores data across multiple storage systems
  - A parallel file system breaks up the data and distributes the blocks to multiple storage servers
  - Designed for high-performance computing systems

- **Why Collecting I/O Statistics?**
  - It is essential for HPC System Administrators to monitor the file systems and storage servers
  - Due to distributed nature of Parallel File Systems, monitoring I/O operations on these systems is complicated
  - That requires a special monitoring systems.

# Collecting I/O Statistics from Lustre Parallel File System

- **Lustre File System**



# Collecting I/O Statistics from Lustre Parallel File System

- **What you learn in this course project?**
  - The structure of Lustre File System and how it works
  - How to read Lustre statistics
  - How to deal with Message Brokers (i.e. RabbitMQ)
  - How to build a distributed system to collect file system statistics

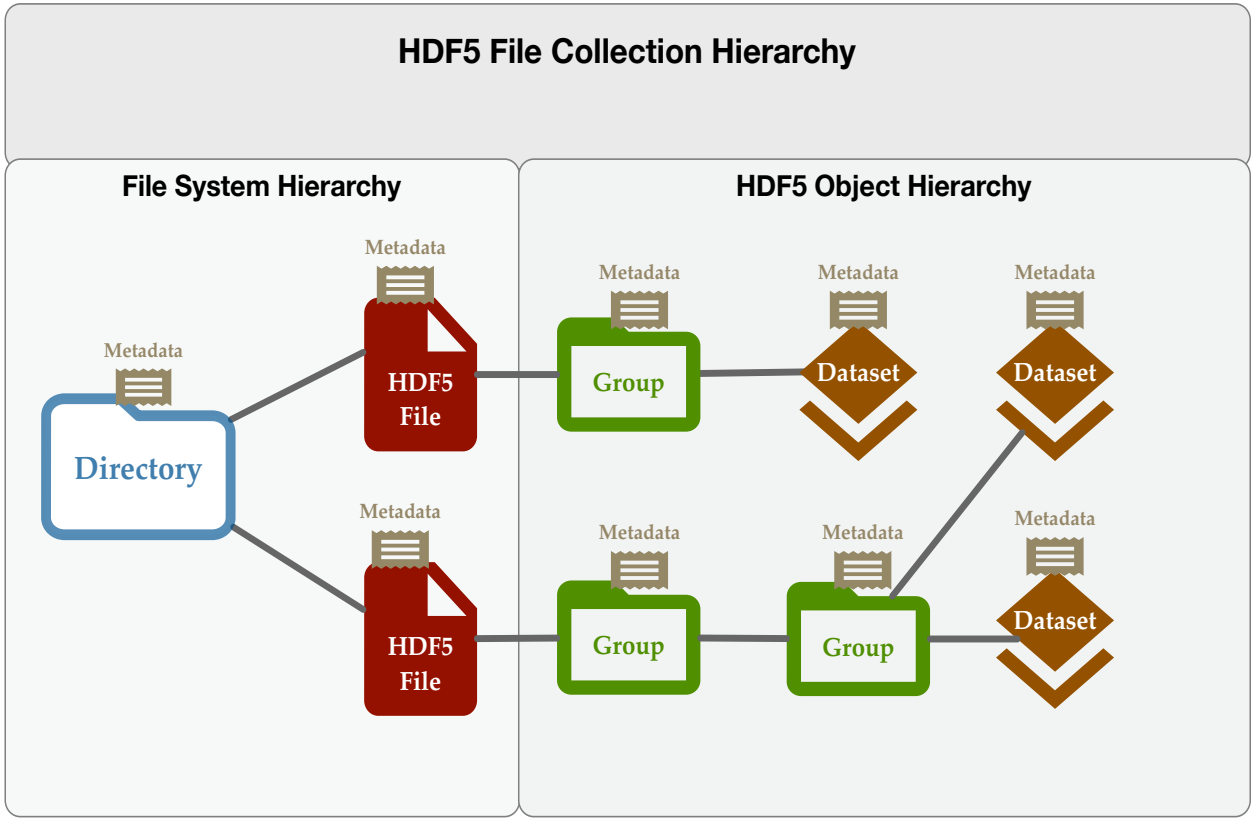
- **What you need to know:**
  - Good knowledge of Linux
  - A programming language (e.g. Python 3.x programming)
  - Database systems (e.g. NoSQL DBs)
  - Work with Virtual Machines

# Topic: #4
# Concurrent and Efficient Metadata Query for Self-describing Data Formats

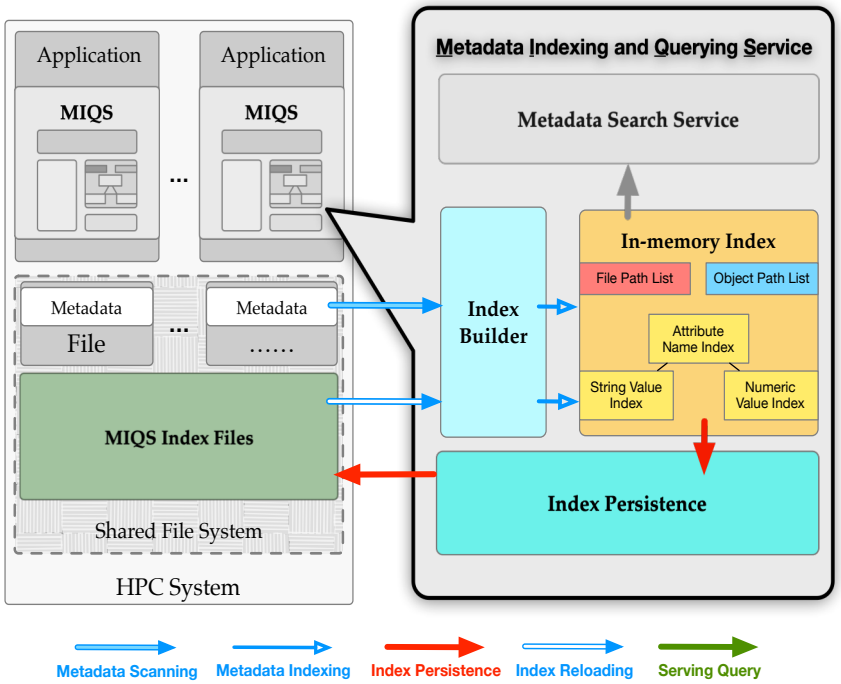
Mr. Wei Zhang
Research Assistant at DISCL
Ph.D. Student in CS

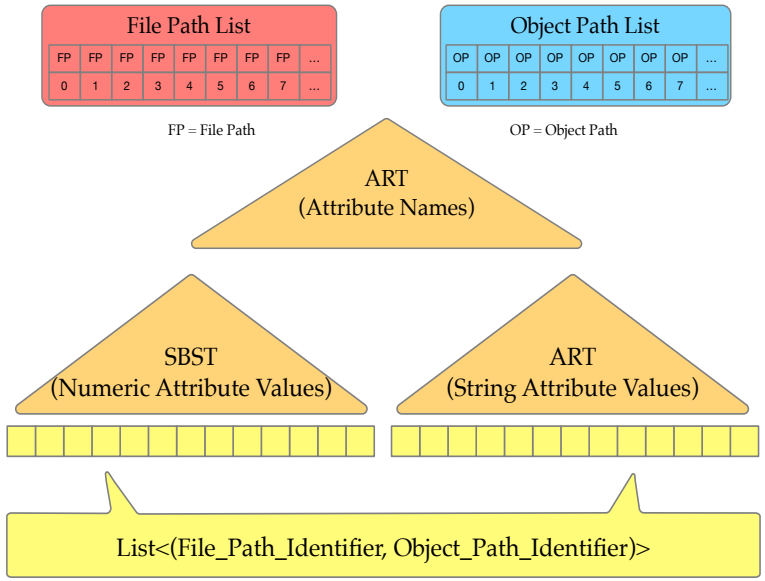
# Concurrent and Efficient Metadata Query for Self-describing Data Formats

## Context: Metadata Search over HDF5 File Collection



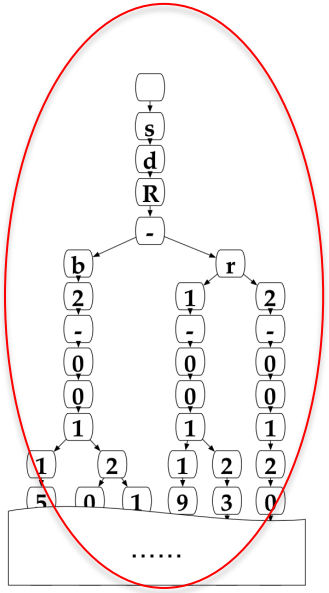
# Concurrent and Efficient Metadata Query for Self-describing Data Formats

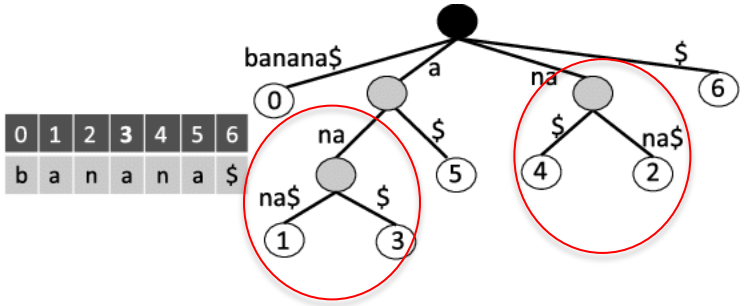


# To be implemented

- Efficient Prefix/Suffix Search
  - Prefix Tree/ART
  - Suffix Tree
- Efficient Infix Search
  - Suffix Array
  - Suffix Tree
- Concurrent Indexing Data Structures
  - Locking on entire data structure
  - Locking on each minimal unit
  - Lock-free solution (Preferred)
- Alternative:
  - Compare all the pattern matching algorithms (KMP, Rabin Karp, FA, BoyerMoore)





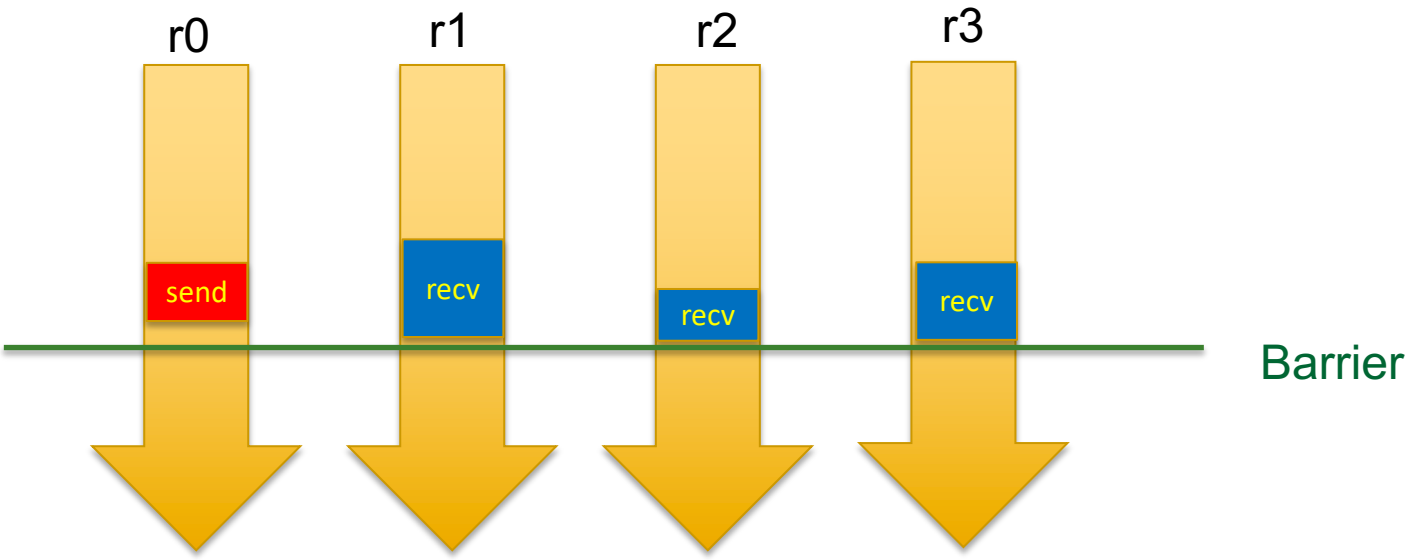
# Readings

- https://dl.acm.org/doi/10.1145/3295500.3356146
- https://github.com/xant/libhl
- https://github.com/armon/libart
- https://people.eecs.berkeley.edu/~stephentu/presentations/workshop.pdf
- https://www.geeksforgeeks.org/pattern-searching-using-suffix-tree/
- https://en.wikipedia.org/wiki/Suffix_array

# Topic: #5
# RPC in MPI-based Programs

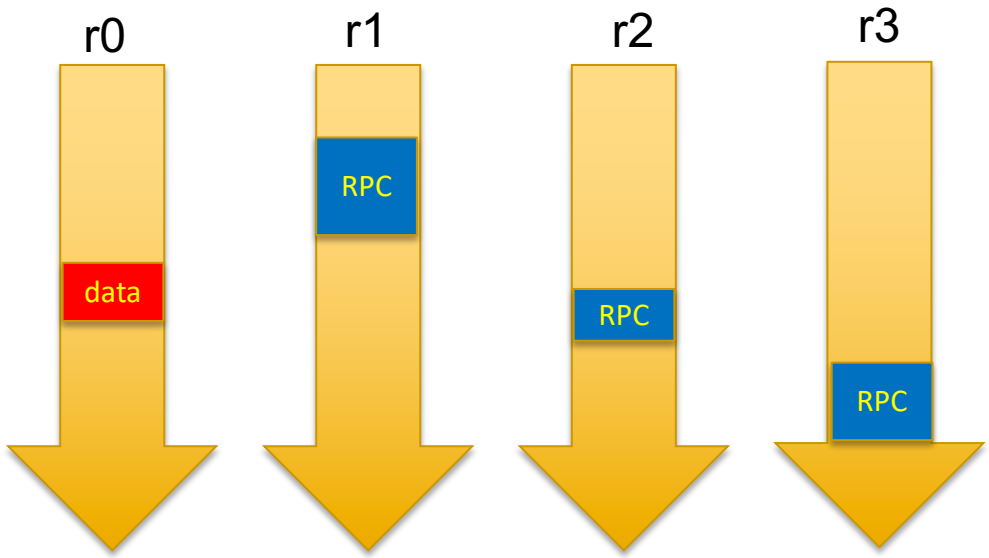
Mr. Wei Zhang
Research Assistant at DISCL
Ph.D. Student in CS

# RPC in MPI-based Programs



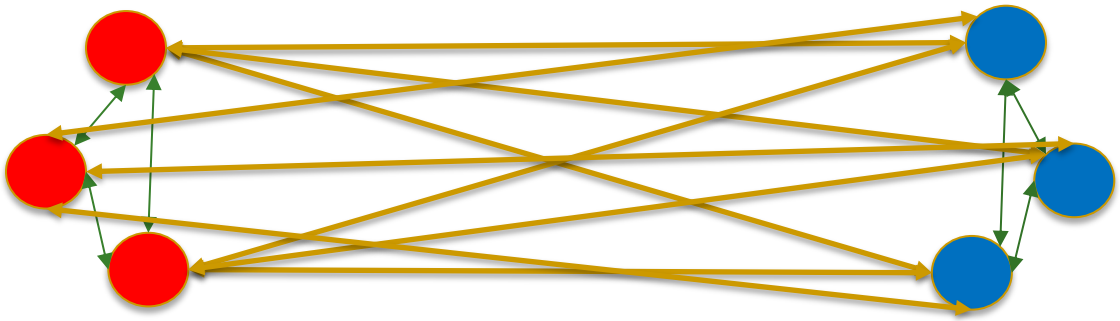

- Data is shared via “push” model
- The data is ready but does not need to be sent immediately.
- The sender has to know where to send, and the process has to be programmed in order to decide whether to receive.
- The developer needs to know how exactly processes coordinate(timing of the data transfer) and which process should be the receiver(counter-intuitive).

# RPC in MPI-based Programs




- Data is shared via “pull” model
- RPC call can take place even when the data is not ready.
- The receiving process just have to issue a RPC call to ask for the data.
- The sender does not have to care where to send. It just has to respond to the RPC call.
- Intuitive programming model.

# To be implemented:



- Write two sets of MPI program.
  - The server-side MPI processes can keep running until a POSIX signal is sent. They can hold a set of data (string and large piece of binary data) when they are running.
  - The client-side MPI processes can issue RPC call to any of the server-side processes and get the requested string or binary data.
  - The server-side MPI processes can communicate with each other using RPC (for both string and binary)
  - The client-side MPI processes can communicate with each other using PRC (for both string and binary)

# Readings

- https://www.open-mpi.org/doc/current/
- https://mercury-hpc.github.io

# Topic: #6
# High Performance Memory Management Library

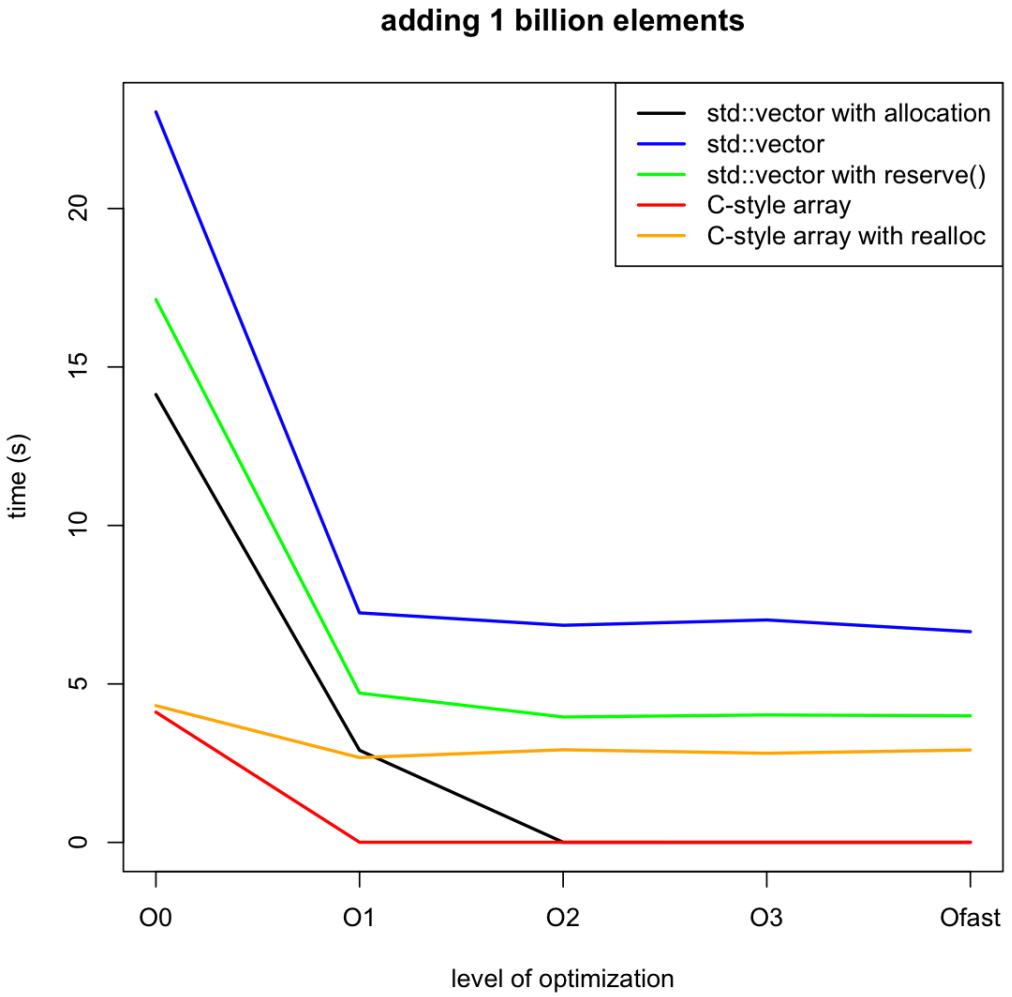
Mr. Xi Wang
Research Assistant at DISCL
Ph.D. Student in CS

# High Performance Memory Management Library

- Applications usually need to dynamically allocate the memory during the program executions due to the limited stack size.
  - C/C++ use malloc() functions to explicitly request memory from OS
  - While Python hides the complexity of memory management with its own interface and implicitly allocate the memory
- However, the memory allocations process involves the operating system kernel to complete, introducing the following overhead:
  - System calls
  - Block the threads
  - Context Switches

# High Performance Memory Management Library

- C++ Vector with default size exhibits highest overhead when pushing 1 Billion data items.
- Allocating a large size in advance greatly reduce the overhead.




https://stackoverflow.com/questions/49615076/is-the-poor-performance-of-stdvector-due-to-not-calling-realloc-a-logarithmic

# High Performance Memory Management Library

- Therefore, it has been a convention that a good software should eliminate the frequency of redundant memory allocations.

- In this project, you will be working on a memory management library
  - implement a lightweight memory management library similar to the Jemalloc for the high performance threading management
  - allocates the memory with disparate sizes in advance and returns a pointer back to the user's allocation request.
  - the allocations are optimized to a simple pointer passing procedure and backend allocated buffer management.

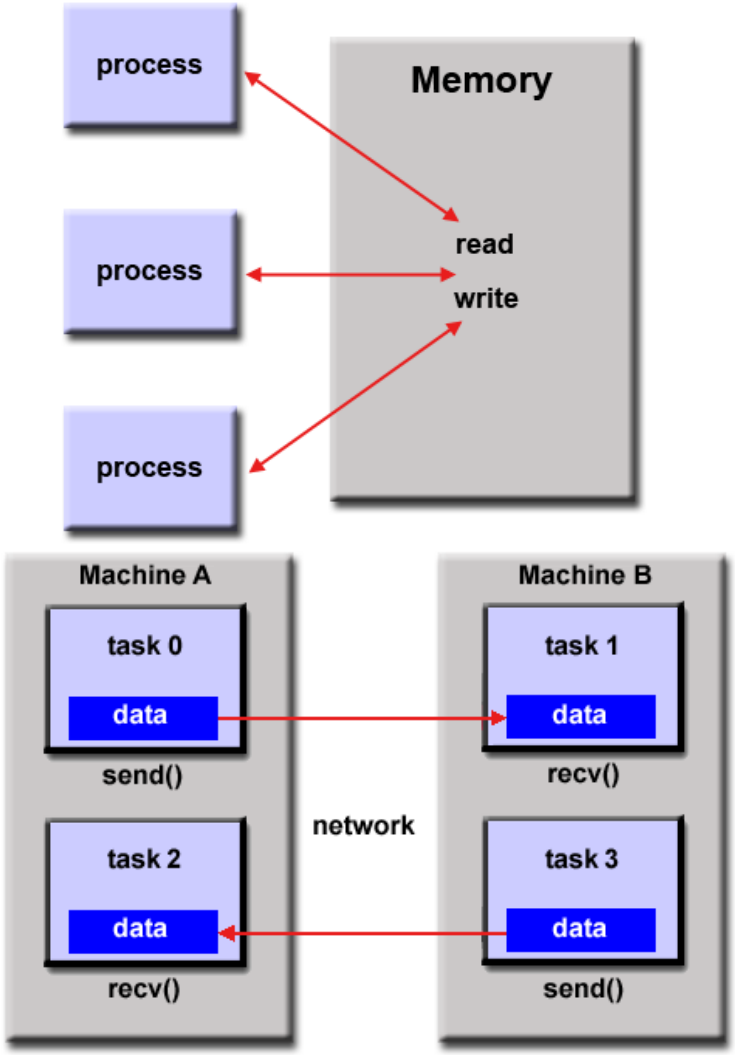
- You can learn the Jemalloc API as an example to see how it works: http://jemalloc.net

# Topic: #7
# A Graph Benchmark for Distributed Memory

Mr. Brody Williams
Research Assistant at DISCL
Ph.D. Student in CS

# A Graph Benchmark for Distributed Memory

- Parallel Programming Paradigms
  - Shared Memory
    - Pthreads
    - OpenMP
    - OpenACC
  - Distributed Memory
    - MPI
    - OpenSHMEM




Images from: https://computing.llnl.gov/tutorials/parallel_comp/

# A Graph Benchmark for Distributed Memory

- OpenMP
  - Both shared and private variables between threads
  - Compiler-based directives
    - #pragma omp XXXX *clauses*
  - Runtime library calls
    - omp_set_num_threads()
- MPI
  - All data is private to each MPI process
  - Two-way message passing for interprocess communication (usually)
- OpenSHMEM
  - All data is private to each OpenSHMEM process
  - One-way communication via gets/puts

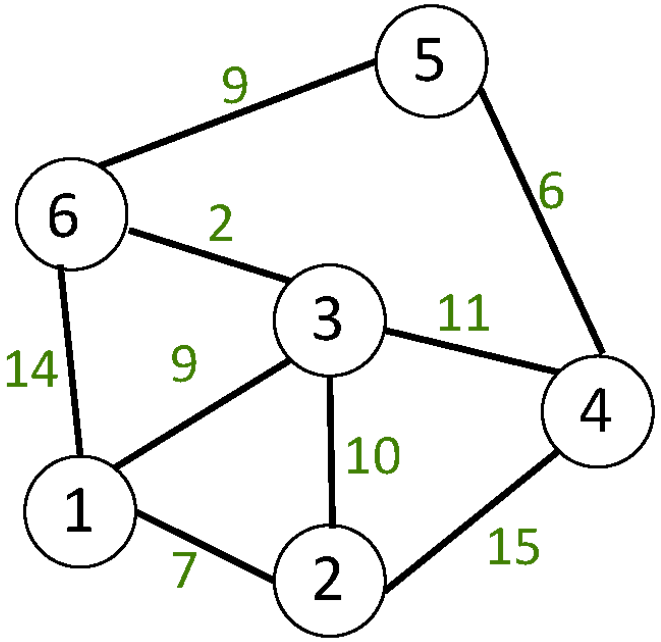
# A Graph Benchmark for Distributed Memory

- Scalable Synthetic Compact Application Suite (SSCA)
  - Designed to simulate the behavior of modern scientific workloads
  - Three different benchmarks
    - Bioinformatics Optimal Pattern Matching
    - Graph Analysis
    - Synthetic Aperture Radar Application
  - Graph Analysis
    - "SSCA#2 is a graph theory benchmark representative of computations in informatics and national security. It is characterized by integer operations, a large memory footprint, and irregular memory access patterns; It is relatively harder to parallelize compared to the other two SSCAs."*
    - Four Kernels
      - Graph construction
      - Find max edge weight
      - Graph extraction
      - Betweenness centrality

* Madduri, Kamesh & Gilbert, John & Shah, Viral & Kepner, Jeremy & Meuse, Theresa. (2006). Designing Scalable Synthetic Compact Applications for Benchmarking High Productivity Computing Systems.

# A Graph Benchmark for Distributed Memory

- Graph Data Structures
  - Vertices/Nodes & Edges
  - Weighted vs Non-weighted
  - Directed vs Undirected
  - Adjacency List




Source: http://homes.sice.indiana.edu/classes/spring2016/csci/c343-yye/undirectedgraph1.png

- Challenges:
  - Partitioning the graph across distributed memory
  - Modifying the computation algorithms accordingly

# Topic: #8
# Performance Analysis of Parallel Applications

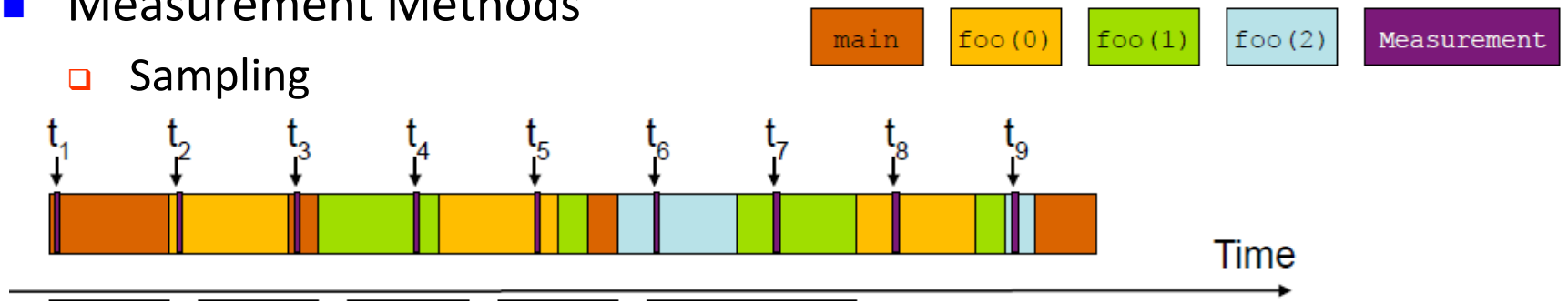
Mr. Brody Williams
Research Assistant at DISCL
Ph.D. Student in CS

# Performance Analysis of Parallel Applications

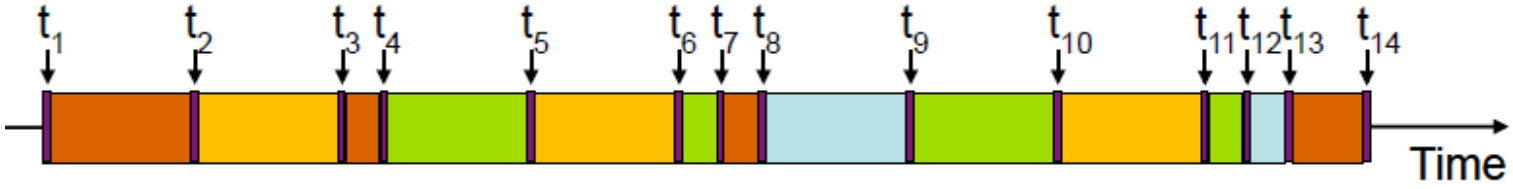
- Performance Analysis Tools
  - HPCToolkit
  - Tuning and Analysis Utility (TAU)
- Visualization Tools
  - Paraprof
  - Vampir
  - Jumpshot
  - Chrome

# Performance Analysis of Parallel Applications

- Profiling
  - Shows how much time is spent in specific code segments
- Tracing
  - Shows when events take place on a timeline
- Measurement Methods
  - Sampling



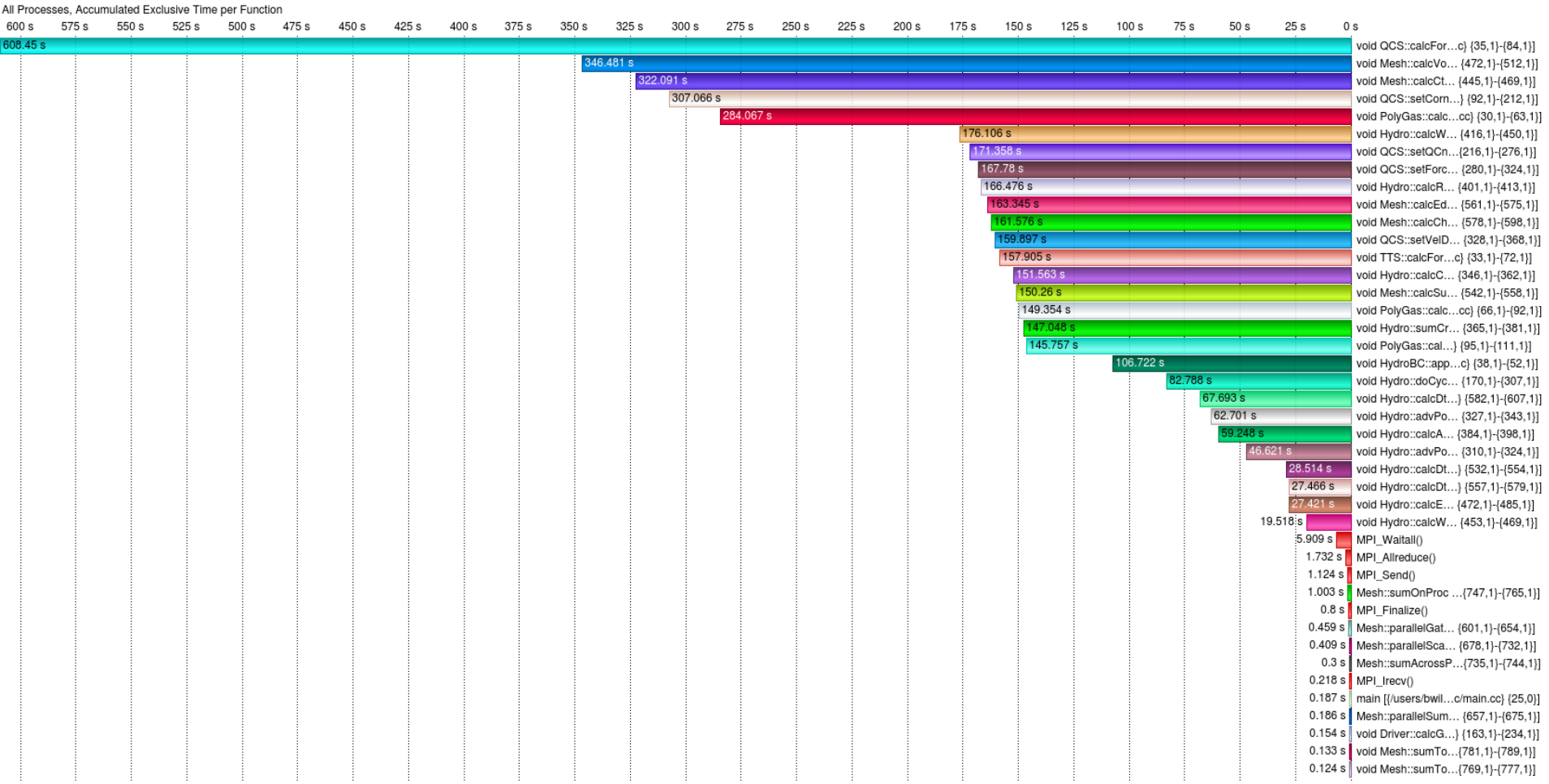

  - Instrumentation




Source: https://www.cs.uoregon.edu/research/tau/tau_summer19.pdf

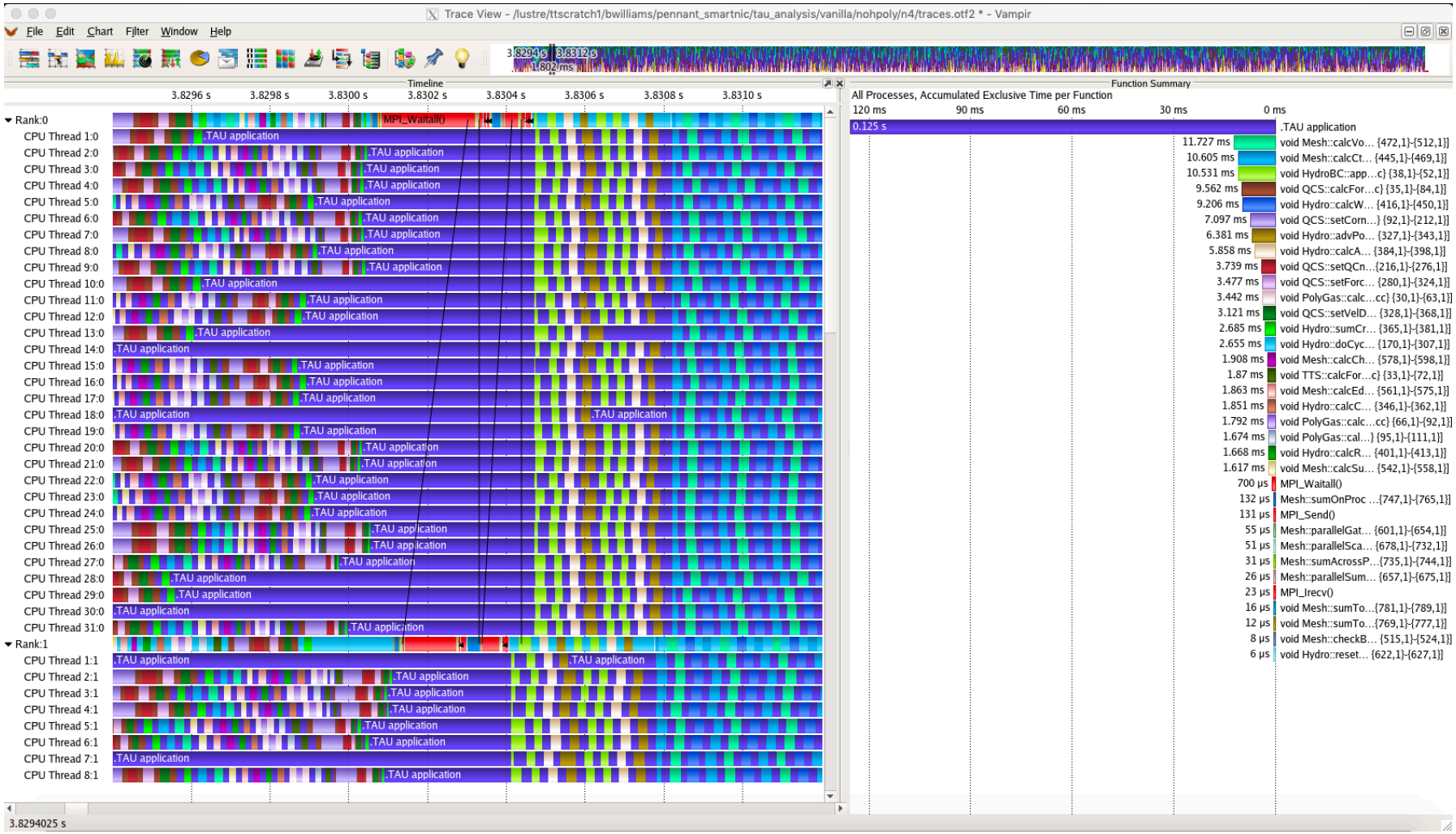
# Performance Analysis of Parallel Applications

## Profiling



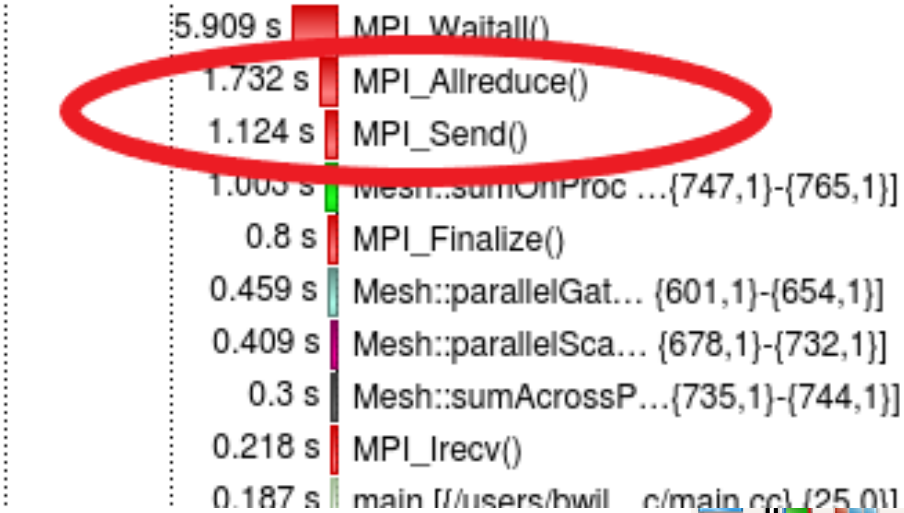
# Performance Analysis of Parallel Applications

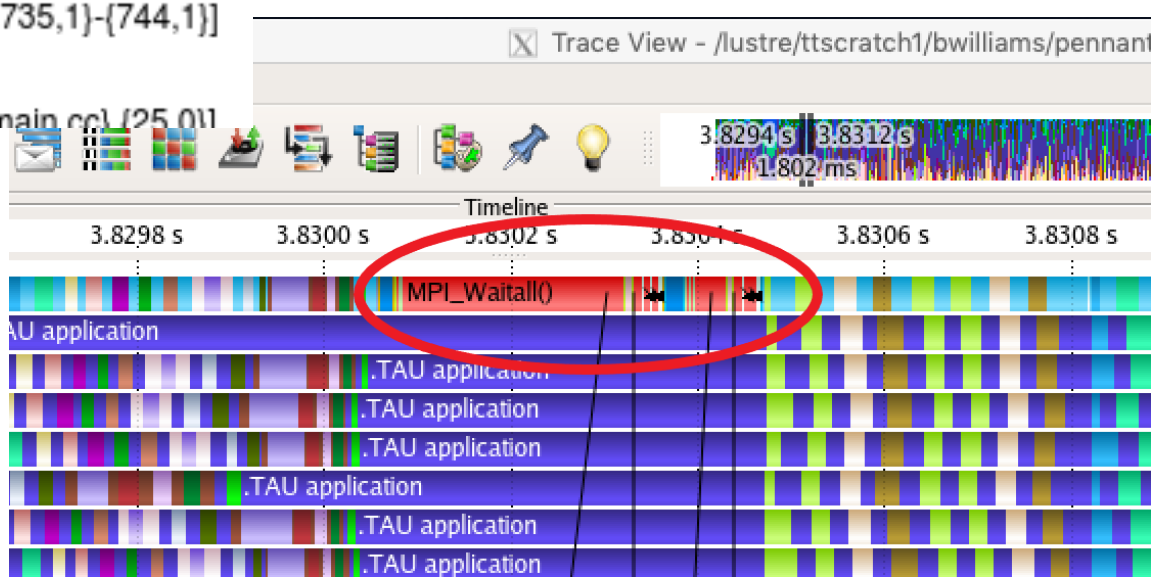
## Tracing



# Performance Analysis of Parallel Applications

## Software-based Communication Overheads



# Topic: #9
# Power Scheduling Schemes for Data Center

Mr. Ghazanfar Ali
Teaching Assistant in CS
Ph.D. Student in CS

# Introduction

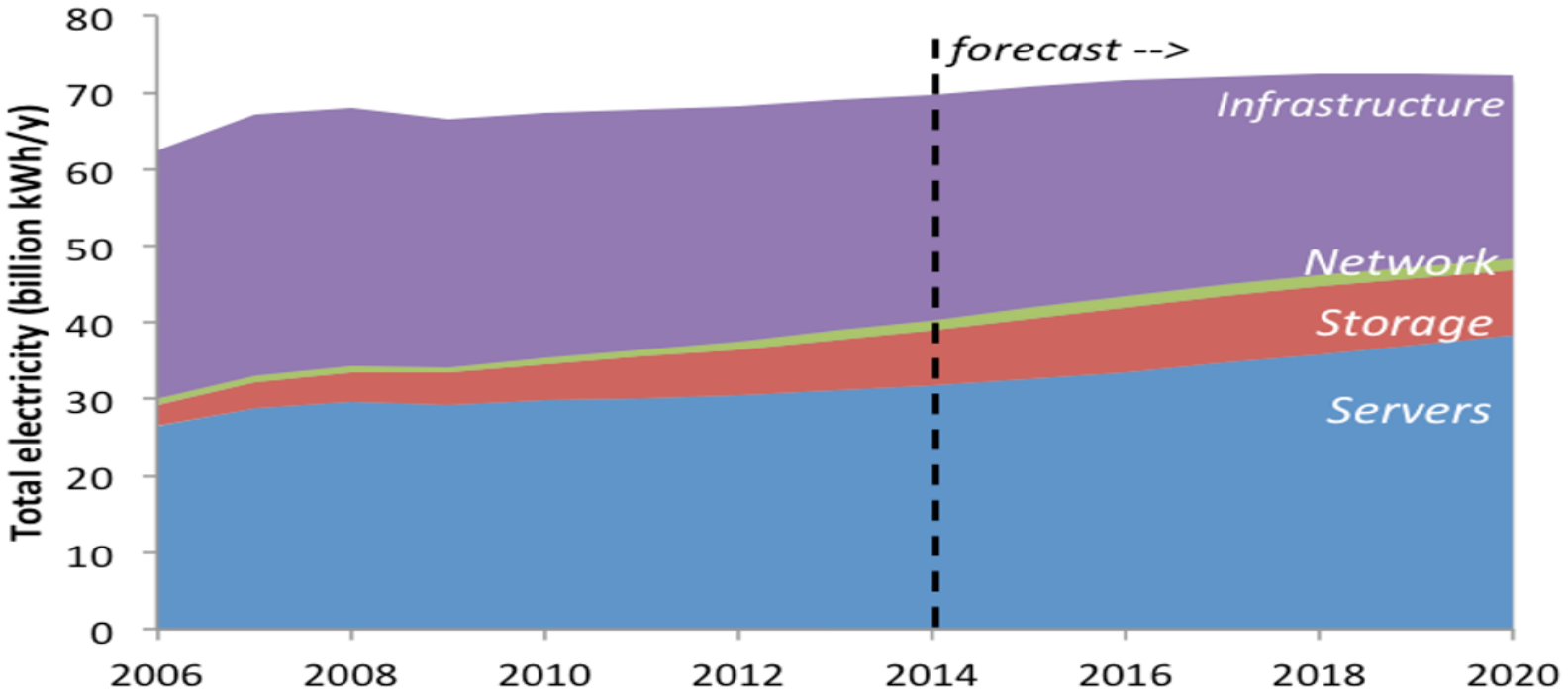
- **Energy**
  - Energy is the capacity to do work.
  - Base Unit is Joule (J)
- **Power**
  - Power is a rate at which **energy** is used or transmitted
  - Base unit is Watts (W) which is Joules per Second

$$P = \frac{\Delta E}{\Delta t}$$

- $P$ is the average power output, measured in watts (W)
- $\Delta E$ is the net change in energy of the system in joules (J).
- $\Delta t$ is the duration - how long the energy use takes - measured in seconds (s)

# U.S. Data Center Energy Consumption (1)

- Data Center Energy Consumption includes servers, storage, network and infrastructure.



# U.S. Data Center Energy Consumption (2)

- U.S. Data Centers Total Energy Consumption: **70** billion kWh/y (~17% )
- Global Data Centers Total Energy Consumption: **416** billion kWh/y

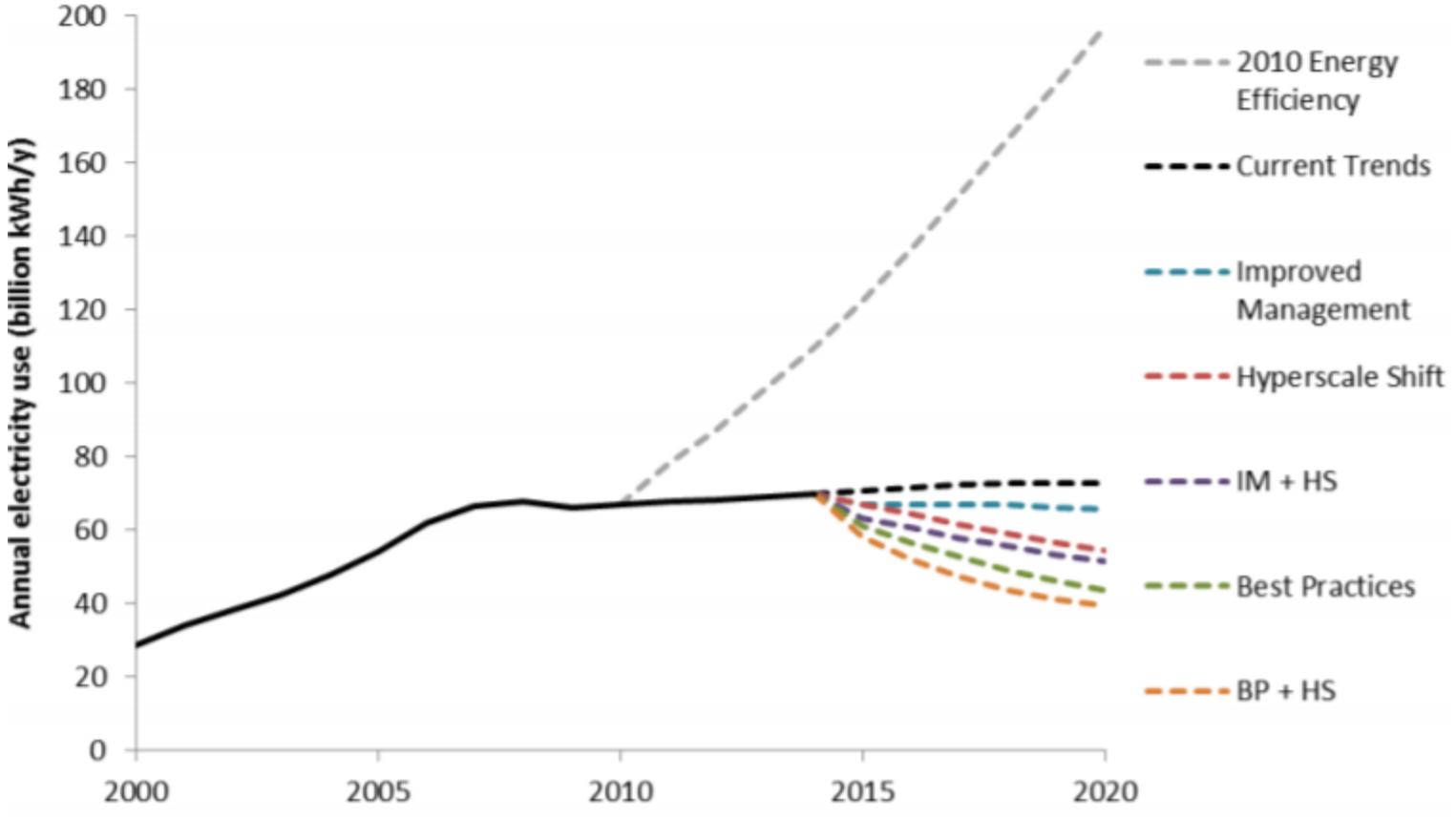
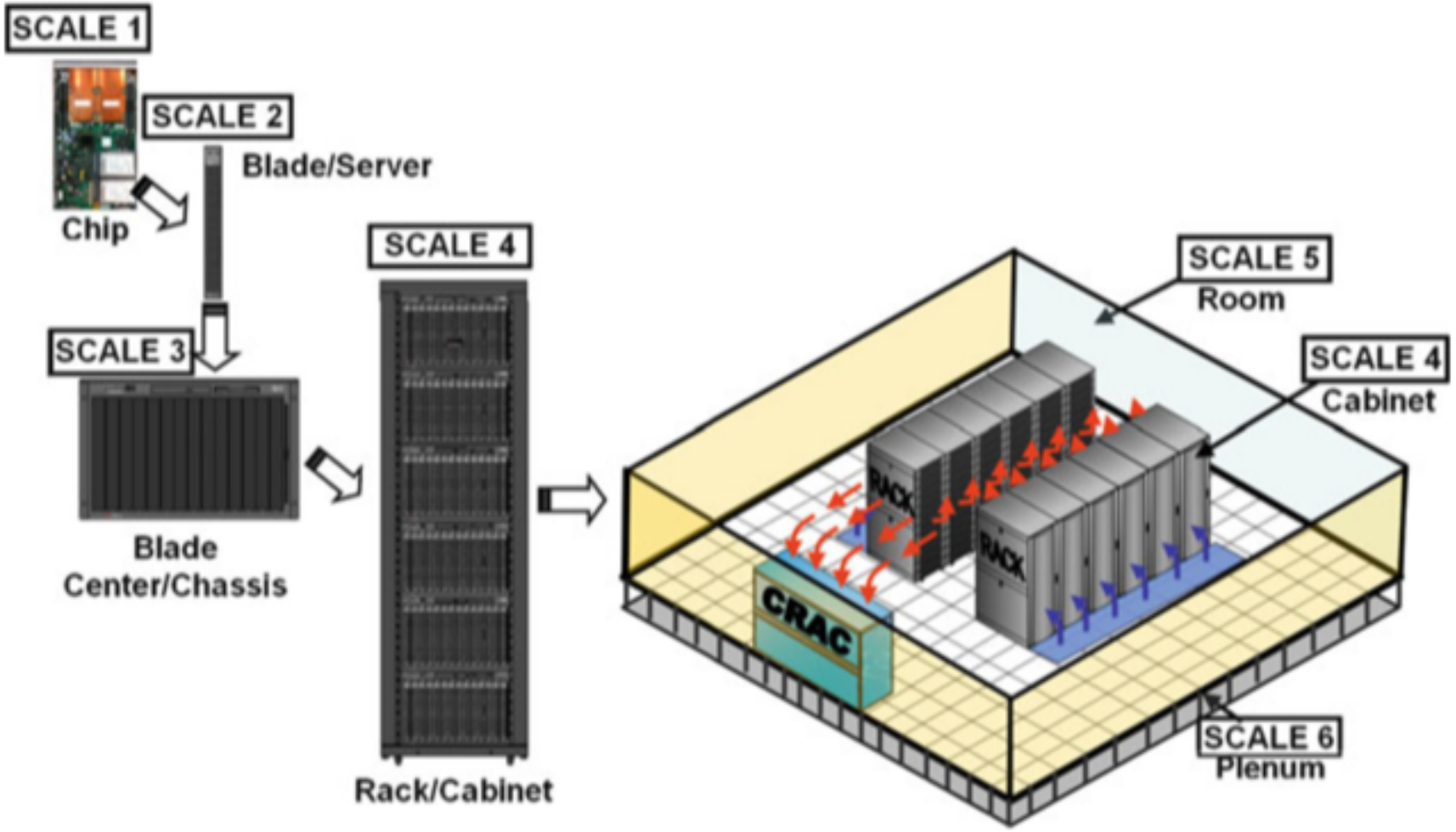



**Figure ES-1 Projected Data Center Total Electricity Use**

https://www.osti.gov/servlets/purl/1372902/

# Granularity in Power Consumption




Source: Energy Efficient Thermal Management of Data Center

# Example of CPU (Chip/Component Level) Power Consumption

- Each tick of the CPU clock (cycle) causes a power consumption
- No. of CPU cycles per second is called CPU frequency (Hz). E.g. one cycle per second is known as 1 hertz.
- This implies that a CPU with a clock speed of 2 gigahertz (GHz) can carry out two billion cycles per second.
- Clocking CPU speed at different CPU frequency have direct impact on CPU power consumption e.g. clocking CPU at 1Ghz will reduce CPU power consumption to (approx.) half.

# Project Description & Deliverables

## High Level Description:

- A data center operator may need to control power consumption.
- Power consumption control could be based on certain allocated power budget.
- Power scheduling methodologies (e.g. power ramping on boot, power scale up and power scale down, reduce active cores) for granular consumption of power in HPC data center.

## Deliverables:

- Identify power on-host power reducing factors/components
- Write power scheduling algorithm
- Operate (ramp up and ramp down) virtual cluster based on power availability (simulated)

# Readings/ToDo

- Knowledge of Golang programming
- Knowledge of virtual clustering (e.g. containers, Kubernetes)

- https://golang.org/doc/

- https://kubernetesbootcamp.github.io/kubernetes-bootcamp/1-1.html

# Topic: #10
# Improving Query Performance of InfluxDB

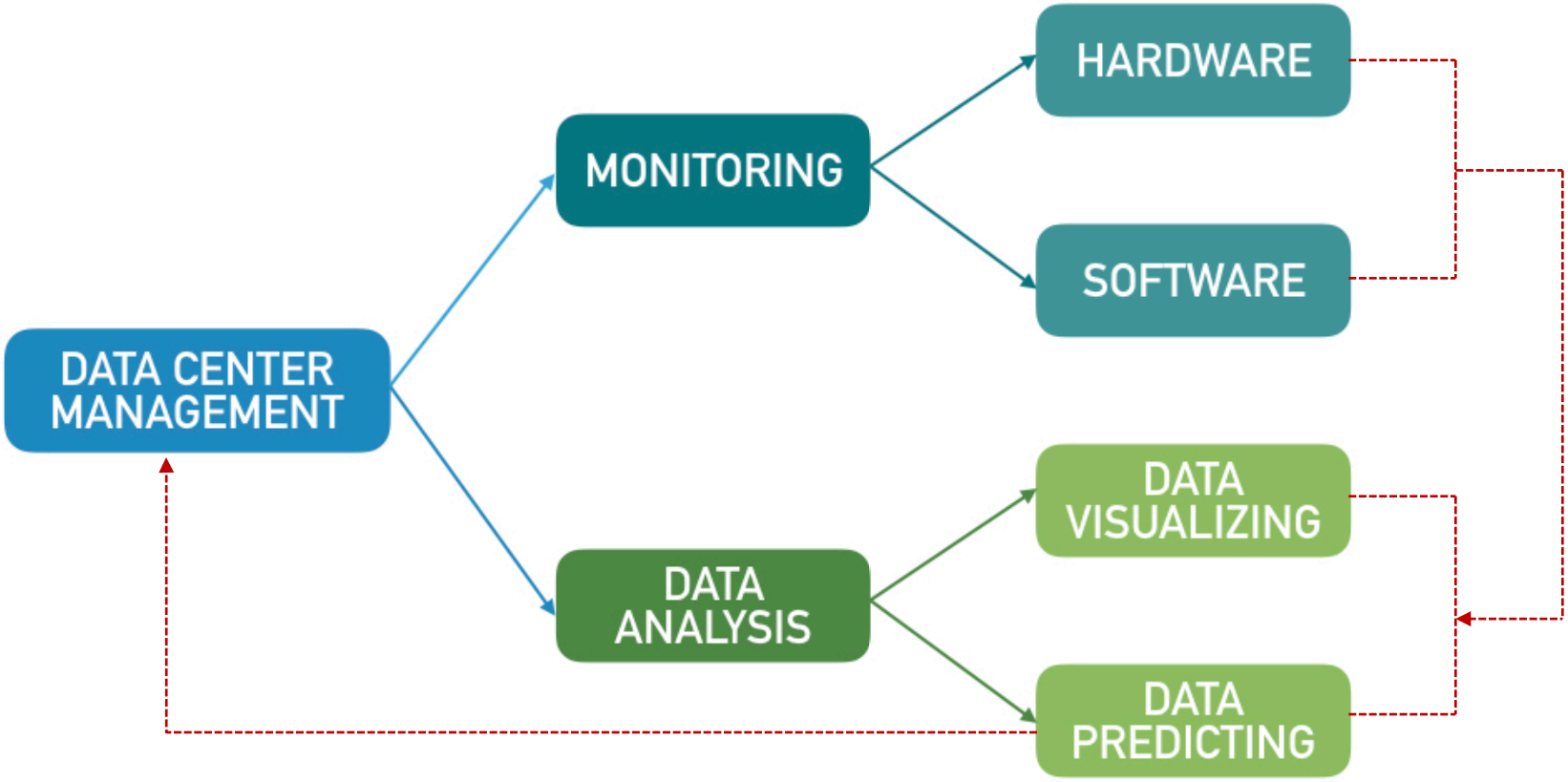
Mr. Jie Li

Research Assistant at DISCL

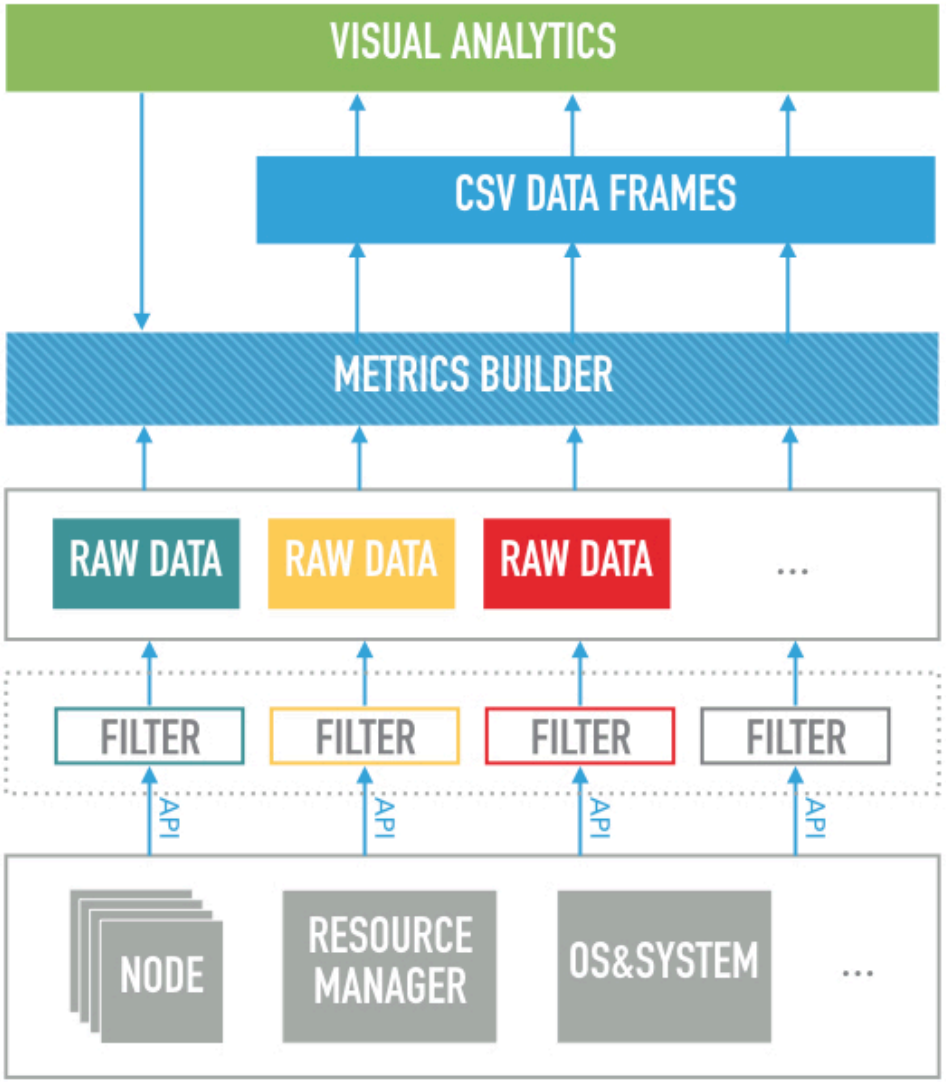
Ph.D. Student in CS

# Data Center Management

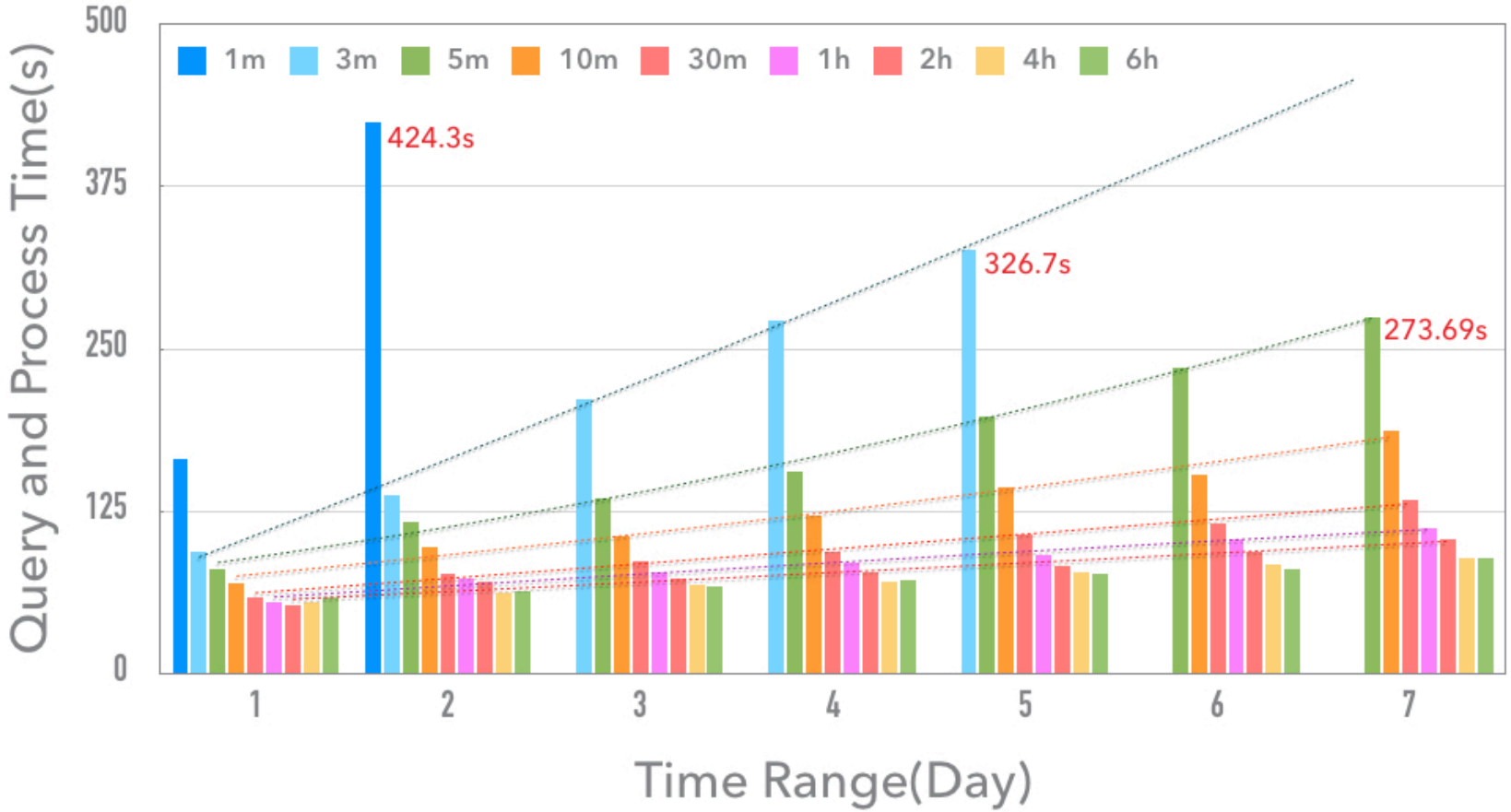
- IMPORVE the way in which systems and applications operate



# Monitoring Framework



# Query Performance



# Research Topic:

- Optimize schema design in our current implementation
- Improve querying performance in influxDB

# Readings

- InfluxDB: https://docs.influxdata.com/influxdb/v1.7/
- MetricsBuilder: https://github.com/nsfcac/MetricsBuilder

# Topic: #11
# Profiling Power Consumption of Jobs on HPC Systems

Mr. Jie Li
Research Assistant at DISCL
Ph.D. Student in CS

# Previous Proposal


- Assumption: power usage is proportional to the core usage
- Assumption is not applied in all situation

# IPMI

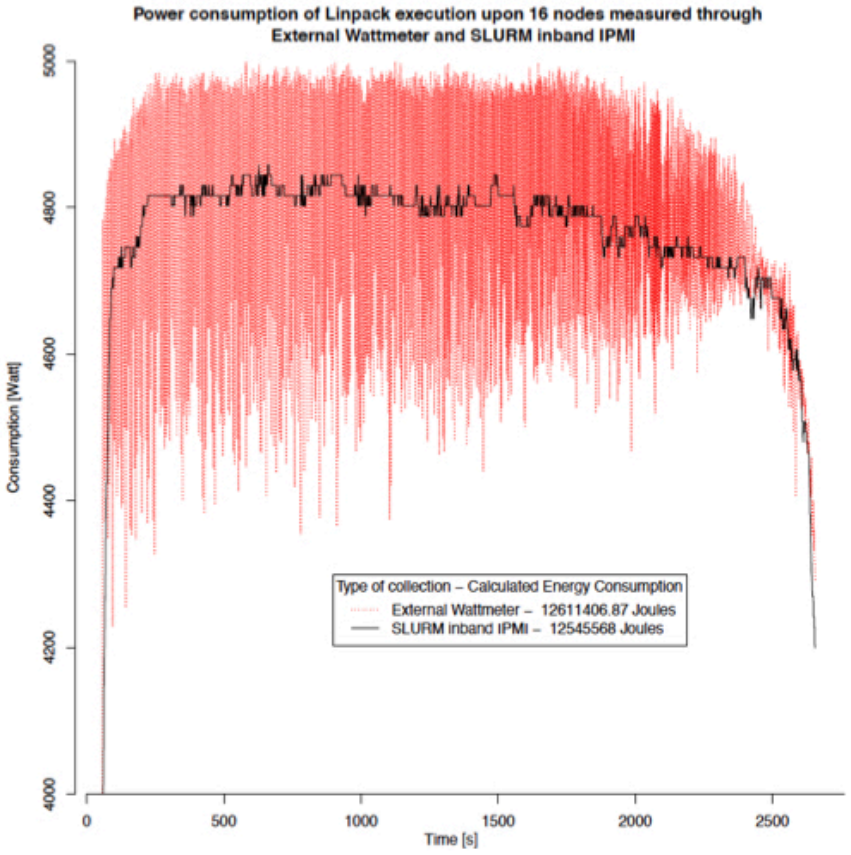
- **I**ntelligent **P**latform **M**anagement **I**nterface(IPMI)
- Message-based, hardware-level interface specification
- Used to perform recovery procedures or monitor platform status (such as temperatures, voltages, fans, power consumption, etc)
- Hidden on the baseboard management controller(BMC) which collects data from various sensors
- Can be found in nearly all current Intel architectures

# RAPL

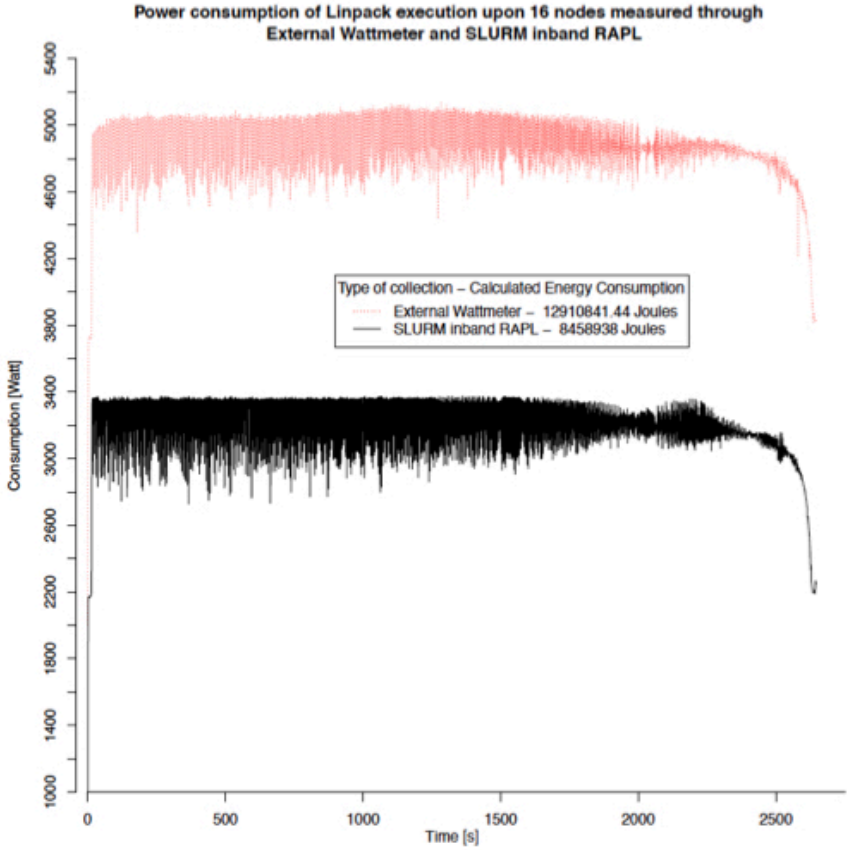
- **R**unning **A**verage **P**ower **L**imit (RAPL)
- Introduced with the Intel Sandy Bridge processors and exits on all later Intel models
- Provides an operating system access to energy consumption information based on a software model driven by hardware counters
- Tracks the energy consumption of CPUs and DRAM but not that of the actually energy of the machine

# IPMI vs RAPL

Benchmark: Linpack




IPMI




RAPL

# Research Topic:

- Collect power consumption data of CPU bound, memory bound, I/O bound benchmarks
- Build a model for estimating power consumption of jobs

# Readings

- Energy Accounting and Control with SLURM Resource and Job Management System: https://link-springer-com.lib-e2.lib.ttu.edu/chapter/10.1007/978-3-642-45249-9_7
- Rapl in action: Experiences in using rapl for power measurements: https://dl-acm-org.lib-e2.lib.ttu.edu/doi/abs/10.1145/3177754
- Validation of Redfish: The Scalable Platform Management Standard: https://dl-acm-org.lib-e2.lib.ttu.edu/doi/abs/10.1145/3147234.3148136

# Topic: #12
# Checkpointing Technology for Parallel Computing

Ms. Elham Hojati
Research Assistant at CAC
Ph.D. Student in CS

# Checkpointing Technology for Parallel Computing

- Parallel programming  and HPC systems
- Failures in supercomputers
- Crash failures
- Error recovery
- Checkpoint-based protocols

# Checkpointing Technologies

- SCR
- DMTCP
- BLCR
- IRIX CPR
- Docker
- Singularity
- Mementos
- Idetic
- …

# Steps

- Provide a survey of checkpointing technologies for parallel computing and HPC Systems.
- Comparing those technologies.
- Implementing one of these checkpointing technologies.

# Implementation

- Chameleon cloud

  https://www.chameleoncloud.org

- Cloud Lab

  https://www.cloudlab.u

# Questions?

## Questions/Suggestions/Comments are always welcome!

Write me: yong.chen@ttu.edu
Call me: 806-834-0284
See me: ENGCTR 315

*If you write me an email for this class, please start the email subject with [CS4379] or [CS5379].*