POLITECNICO
MILANO 1863

HPC software, relevant qualities and SE methods

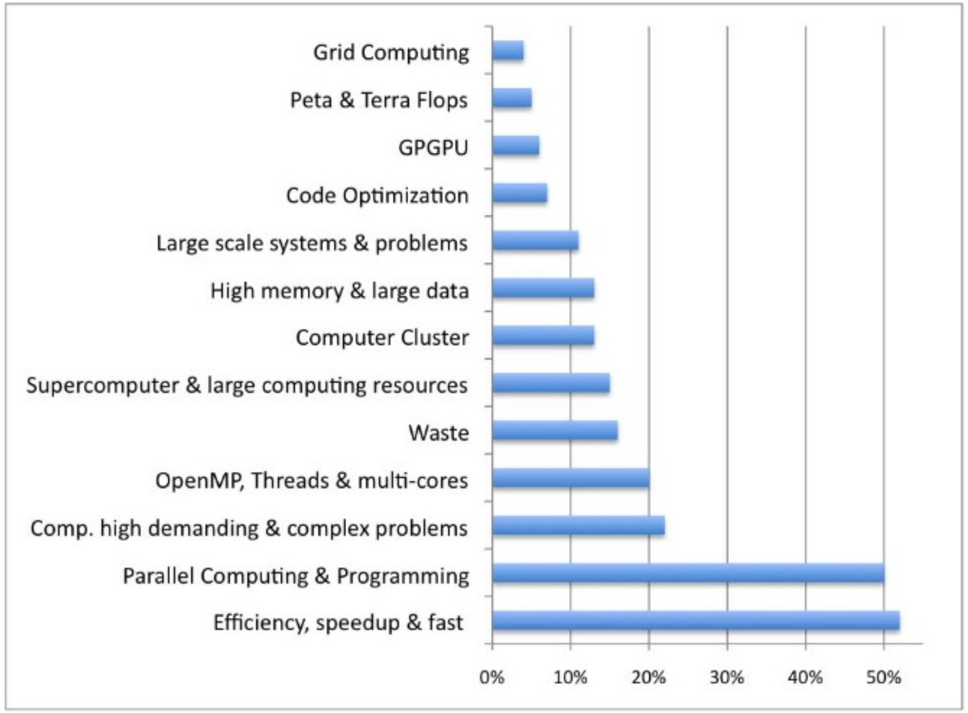
“Assignment” for next lecture (Wednesday)

- Prepare your own answers to the following questions
    - What is High Performance Computing?
    - What are the qualities a HPC software should offer?
    - What are the most critical issues we should pay attention to during the lifecycle of HPC software?

HPC - definitions

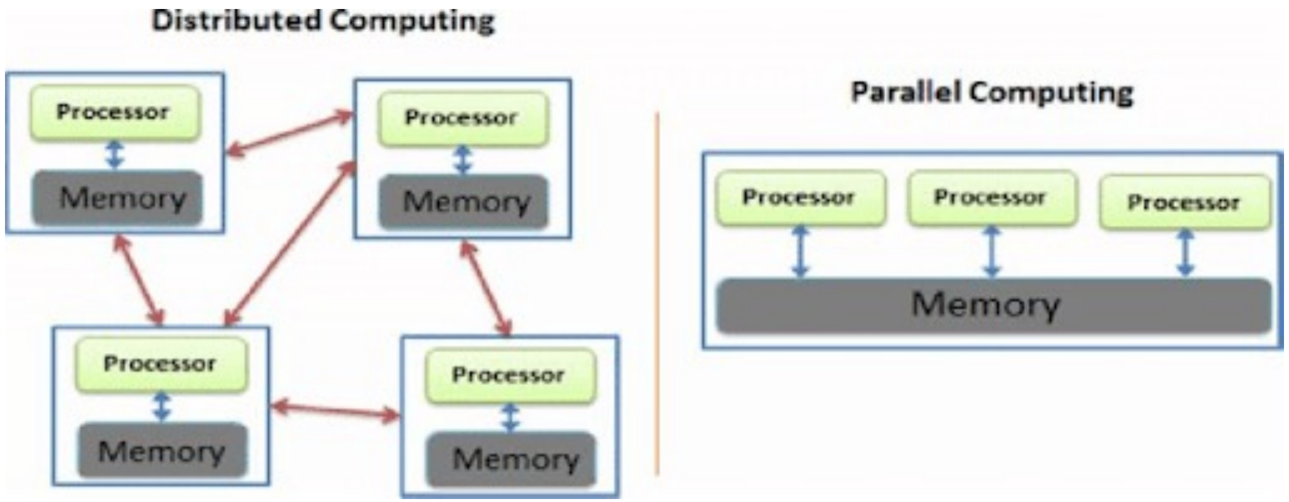
- From [6]
  - The practice of aggregating computing power in a way that delivers much higher performance than one could get out of a typical desktop computer or workstation to solve large problems in science, engineering, or business
  - Thousands of processors working in parallel to analyse billions of pieces of data in real time, performing calculations thousands of times faster than a normal computer
  - The use of parallel processing for running advanced, large-scale application programs efficiently, reliably and very quickly on supercomputer systems
  - The platform technology concerned with programming for performance, where performance takes the broad meaning of speed (reducing time to solution), energy efficiency (doing more with less power), upscaling (handling larger problems such as simulating a wing and then a full plane, or a cell and then an organ) or high throughput (the ability to handle large volumes of data in near real-time, as required in the financial services industry, telecoms or satellite imagery)
- From [1]
  - Use of cutting-edge computing technology, both hardware and software, to solve otherwise impossible problems

Relevant characteristics of HPC results from a survey [1]

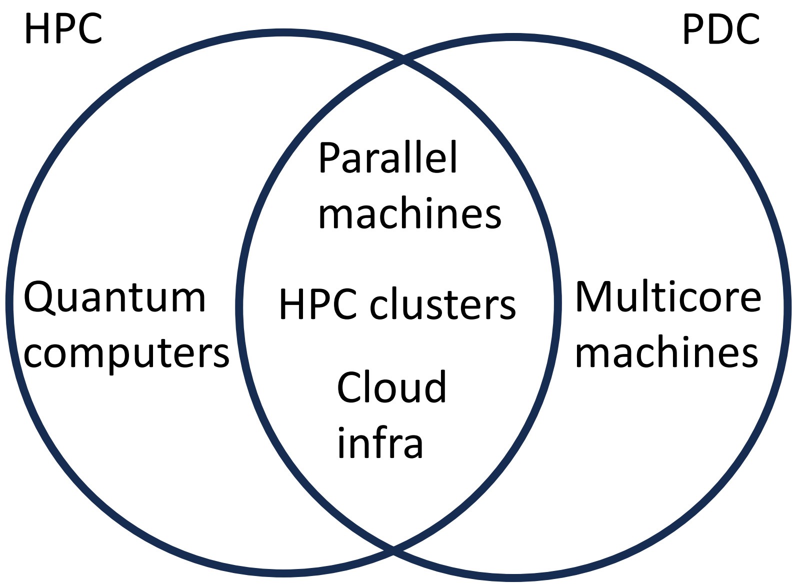


Parallel and Distributed Computing (PDC)

- Main characteristics
  - Concurrency: property of software
    - A piece of software is concurrent if it may have more than one active execution context
  - Parallelism: property of hardware
    - Execution of different tasks/pieces of software at the same time
  - Distribution
    - Execution of different tasks/pieces of software on physically distinct computing nodes connected through a network,
      lack of a global clock



HPC vs PDC

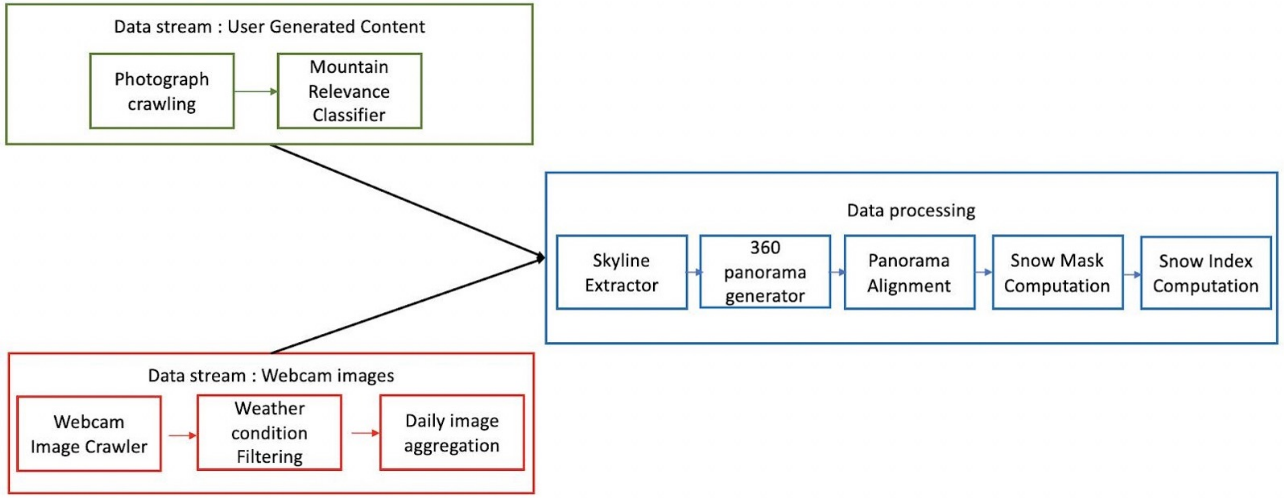

HPC
PDC
Quantum computers
Parallel machines
HPC clusters
Cloud infra
Multicore machines

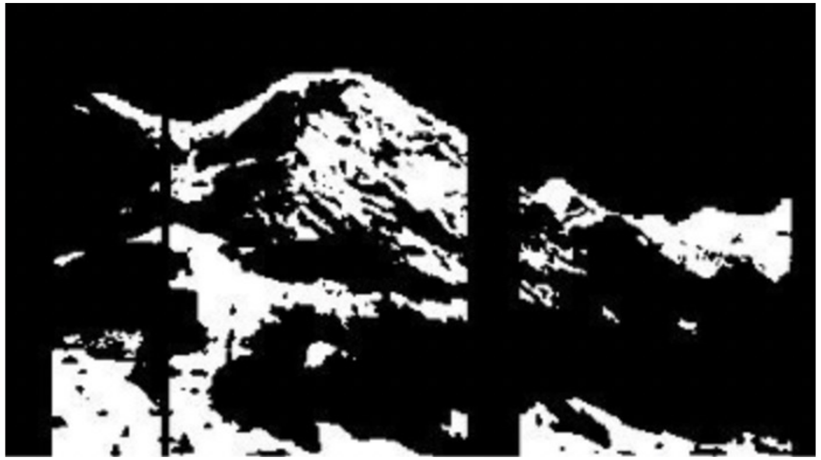
Cathegories of HPC software

- Compute-intensive applications
  - Concern complex computations requiring a large number of computational resources
  - Often require parallel computing

- Data-intensive applications
  - Focus on processing, storing and retrieving large amounts of data
  - Typically, built as distributed systems to ensure reliability and scalability

Example


Data stream : User Generated Content
Photograph crawling
Mountain Relevance Classifier
Data processing
Skyline Extractor
360 panorama generator
Panorama Alignment
Snow Mask Computation
Snow Index Computation
Data stream : Webcam images
Webcam Image Crawler
Weather condition Filtering
Daily image aggregation

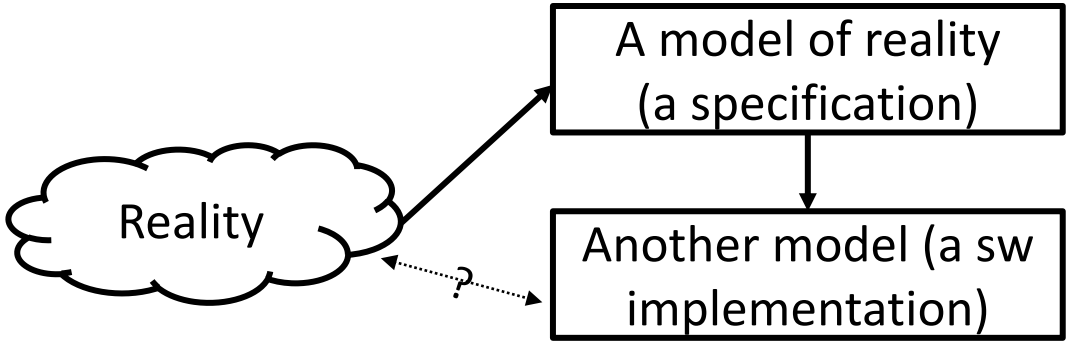


Important qualities for HPC software

- For compute-intensive software [1]
  - Correctness
  - Performance
  - Portability
  - Maintainability
- For data-intensive software [2]
  - Reliability
  - Scalability
  - Maintanability

Correctness

- Software is correct if it satisfies the specifications
- Warning!



- Issue [3]
    - Sometimes difficult/impossible to show actual correctness with respect to reality
        - Imagine you are building a simulator of some planet lander before having ever visited it…

Correctness

- What do we do then?
  - Check whether the software output fulfills the important desired properties
  - Identify a measure of precision and apply it

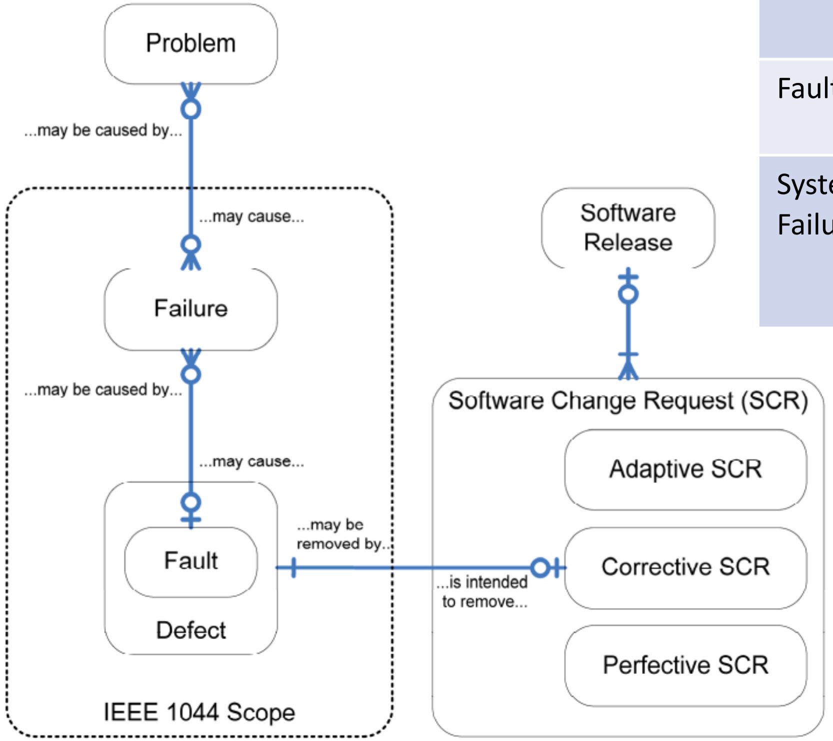
Performance

- Efficient use of resources
- Warning [3]!
    - Is performance improvement always a good idea?
    - Not necessarily if it makes software more difficult to read and maintain!
    - Or if it reduces portability of software

Reliability

- Can be defined mathematically as "probability of absence of failures for a certain time period"
- Typical expectations [2]:
  - The application performs the expected function
  - It can tolerate mistakes by users
  - It prevents unauthorized access and abuse

Software failures, faults, and defects [4]



Term	Description
Defect	An imperfection or deficiency in a work product where that work product does not meet its requirements or specifications and needs to be either repaired or replaced.
Fault	A defect encountered during software execution is a fault (a fault is a subtype of defect)
System Failure	(A) Termination of the ability of a product to perform a required function or its inability to perform within previously specified limits. (B) An event in which a system or system component does not perform a required function within specified limits.

Fault-tolerant systems and reliability

- System that cope with faults and avoid the occurrence of failures are called **fault-tolerant** or **resilient**
- Fault tolerance increases reliability

Hw vs sw faults (1)

- Hardware faults
  - In a large datacenter these can happen on a daily basis
  - Different pieces of hardware usually fail independently from each other
  - Solutions
    - Hardware redundancy
    - Software fault-tolerance techniques

Hw vs sw faults (2)

- Software faults
  - They result from sw development errors
  - Can stay dormant for a long time and appear suddently
  - They can determine failures in multiple components at the same time
- Solutions -> no silver bullets! Adoption of a combination of strategies:
  - Adopt defensive programming
  - Through testing before release
  - During operation
    - Restart the system frequently (rejuvenation)
    - Continuous monitoring and alerting in case of possible sympthoms
    - Deliverately introducing faults to exercise the fault-tolerance machinery (chaos engineering)

Configuration errors

- In [5] it is observed that:
  - Operator errors are the leading cause of failure in Internet services
  - Configuration errors are the largest category of operator errors
  - More extensive online testing and more thoroughtly exposing and detecting component failures would reduce failure rate

Scalability

- System ability to cope with increased load
- How do we make a system scalable?

Load

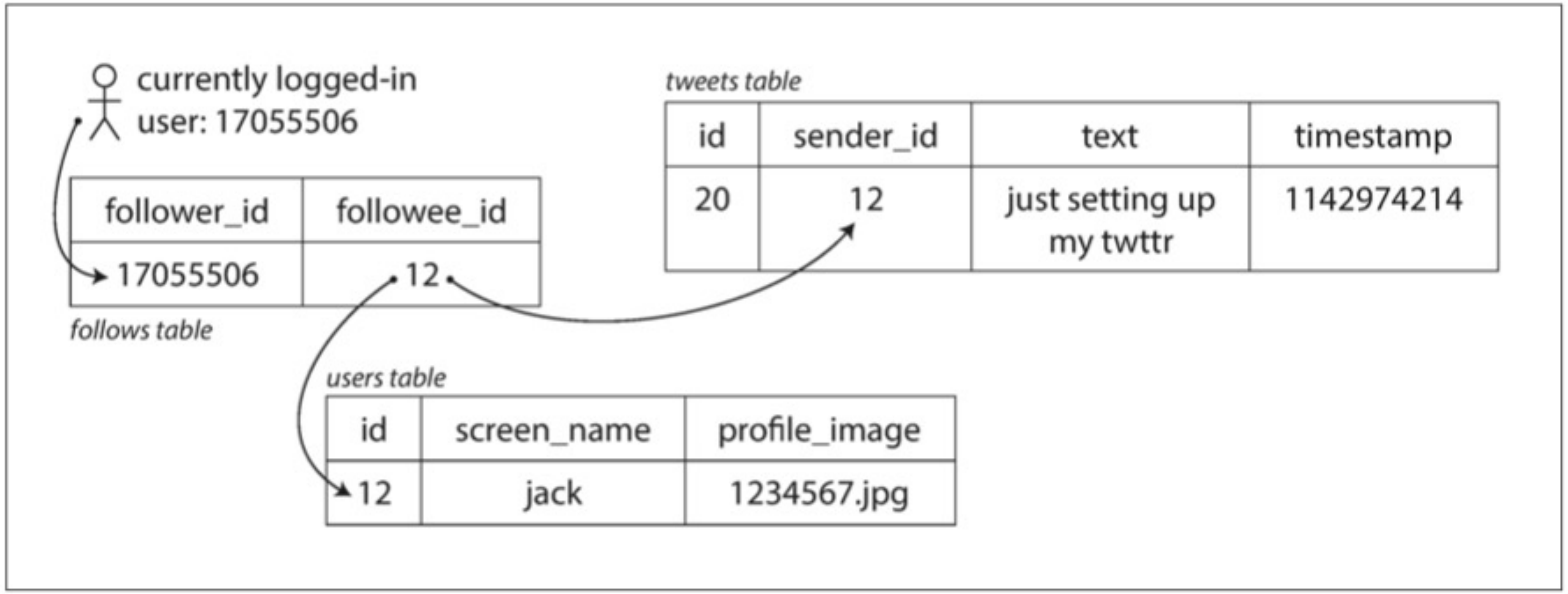
- How can this be represented?
    - Number of requests per second -> suitable for web apps
        - Average number? Number at the peak?
    - Number or read and write operations (or their ratio) -> suitable for databases
    - Number of connected users in a virtual room
    - …

Twitter example

- Two main operations
  - Post tweet: a user can publish a message
    - Data from 2012: 4.6k requests/s on average, 12k requests/s at peak
  - Home timeline: a user can view the twits posted by the people they follow
    - 300k requests/s
- Scalability challenge: cope with the number of connections between followers and followee.

Twitter example [2] – design approach 1

- Tweets are stored in a tweets table
- Any time a user requests the home timeline, we look up at the followed people and retrieve their tweets



currently logged-in user: 17055506

follows table

follower_id	followee_id
17055506	12

tweets table

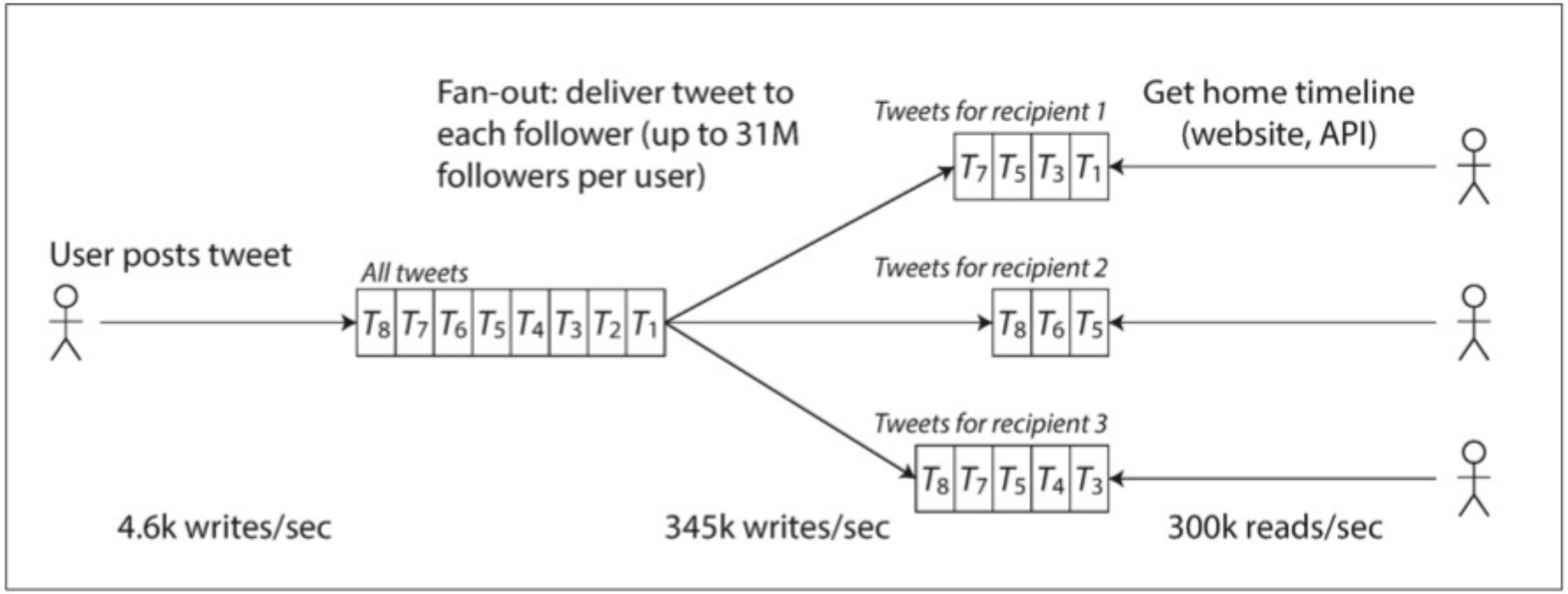
id	sender_id	text	timestamp
20	12	just setting up my twttr	1142974214

users table

id	screen_name	profile_image
12	jack	1234567.jpg

Twitter example [2] – design approach 2

- Tweets from users with a large number of followers are replicated in the tweets sets relevant for each follower
- Tweets are immediately available when the followers require them


Fan-out: deliver tweet to each follower (up to 31M followers per user)
Get home timeline (website, API)
User posts tweet
All tweets
T_8 T_7 T_6 T_5 T_4 T_3 T_2 T_1
Tweets for recipient 1
T_7 T_5 T_3 T_1
Tweets for recipient 2
T_8 T_6 T_5
Tweets for recipient 3
T_8 T_7 T_5 T_4 T_3
4.6k writes/sec
345k writes/sec
300k reads/sec

Maintainability

- Three principles to follow
  - Operability: make it easy for the operation team to run the system and keep it running
  - Simplicity: make it easy for other software engineers to understand the system
  - Evolvability: make it easy for engineers to change the system when a new requirement emerges

Operability

- Possible actions
  - Providing visibility into the runtime behavior and internals of the system, with good monitoring
  - Providing good support for automation and integration with standard tools
  - Avoiding dependency on individual machines (allowing machines to be taken down for maintenance while the system as a whole continues running uninterrupted)
  - Providing good documentation and an easy-to-understand operational model (“If I do X, Y will happen”)
  - Providing good default behavior, but also giving administrators the freedom to override defaults when needed
  - Self-healing where appropriate, but also giving administrators manual control over the system state when needed

Simplicity

- Complex systems require more time to be understood and increase the cost of maintenance
- Possible actions
  - Reduce *accidental complexity*
  - Use abstractions
    - Organize the architecture in well-defined components that hide the internal complexity behind a clear and simple to use interface
    - Reuse known solutions

Evolvability

- Possible actions
  - Organize your development process to cope with evolution
  - Keep track of how requirements map into your software structure
  - Update documentation
  - Continue to ensure simplicity and operability

What are the SE methods needed in HPC?

- From [1] and [6]
  - Modelling the software structure and checking its properties
  - Performance analysis and improvement
  - Source code management
  - Documentation, standards, support to maintainability
  - Support to scalability
  - Attention to operability and automation

Bibliography

- [1] M. Schmidberger and B. Brügge, "Need of Software Engineering Methods for High Performance Computing Applications," 2012 11th International Symposium on Parallel and Distributed Computing, Munich, Germany, 2012, pp. 40-46, doi: 10.1109/ISPDC.2012.14.
- [2] Martin Kleppmann, “Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems”, O’Reilly, 2017 (available for Polimi users here https://ebookcentral.proquest.com/lib/polimi/detail.action?docID=4825244 . Chapter 1 available on WeBeep.
- [3] V. R. Basili et al., "Understanding the High-Performance-Computing Community: A Software Engineer's Perspective," in IEEE Software, vol. 25, no. 4, pp. 29-36, July-Aug. 2008, doi: 10.1109/MS.2008.103.
- [4] "IEEE Standard Classification for Software Anomalies," in IEEE Std 1044-2009 (Revision of IEEE Std 1044-1993) , vol., no., pp.1-23, 7 Jan. 2010, doi: 10.1109/IEEESTD.2010.5399061.
- [5] David Oppenheimer, Archana Ganapathi, and David A. Patterson, Why Do Internet Services Fail, and What Can Be Done About It?, 4th USENIX Symposium on Internet Technologies and Systems (USITS 03). 2003
- [6] Rajendra K. Raj et al., “High Performance Computing Education: Current Challenges and Future Directions”. In Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education (ITiCSE-WGR '20). Association for Computing Machinery, New York, NY, USA, 51–74. https://doi.org/10.1145/3437800.3439203