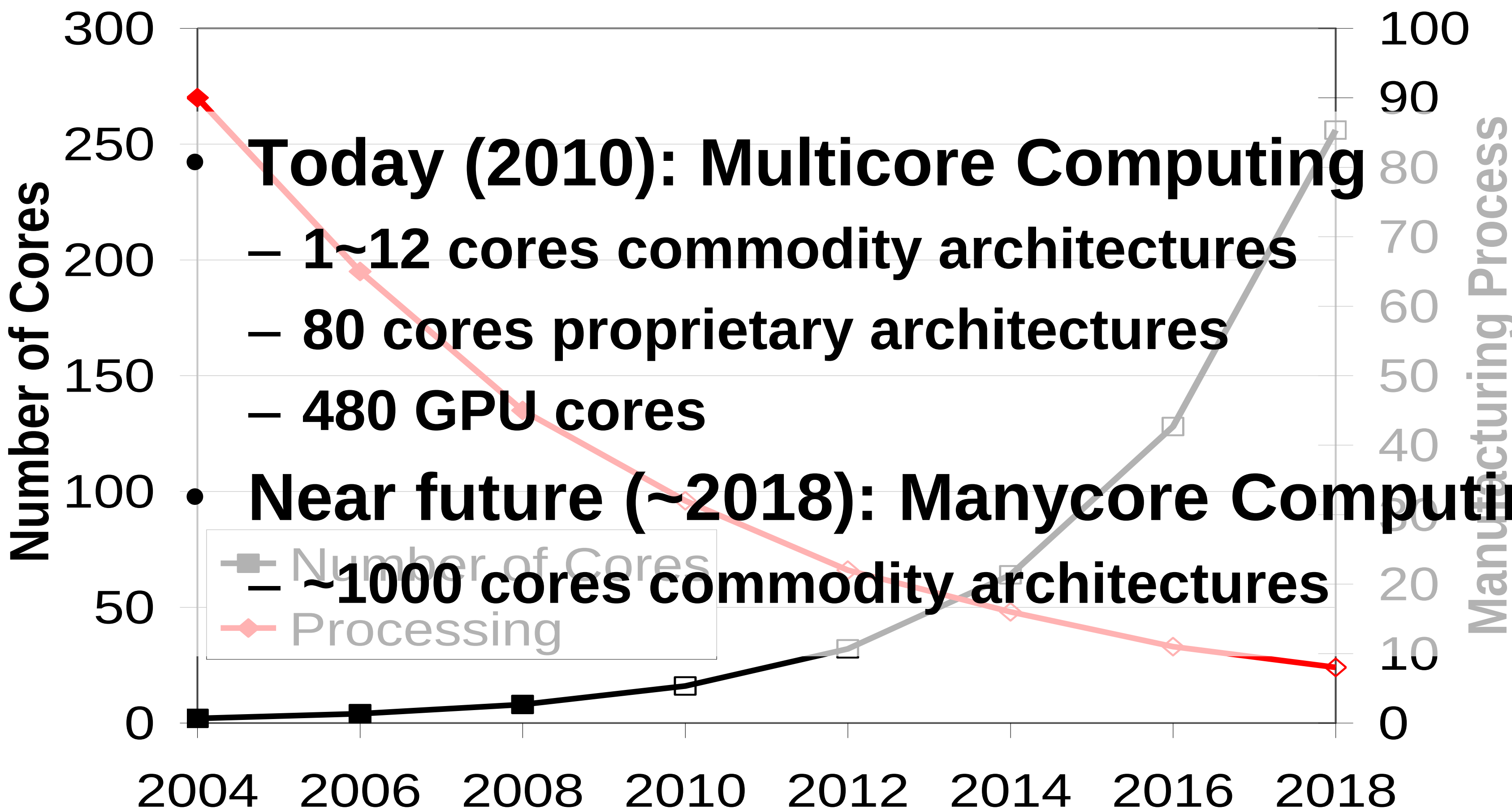
Avoiding Achilles' Heel in Exascale Computing with Distributed File Systems

Ioan Raicu
Computer Science Department, Illinois Institute of Technology
Math and Computer Science Division, Argonne National Laboratory

DLS Seminar
January 14th, 2011

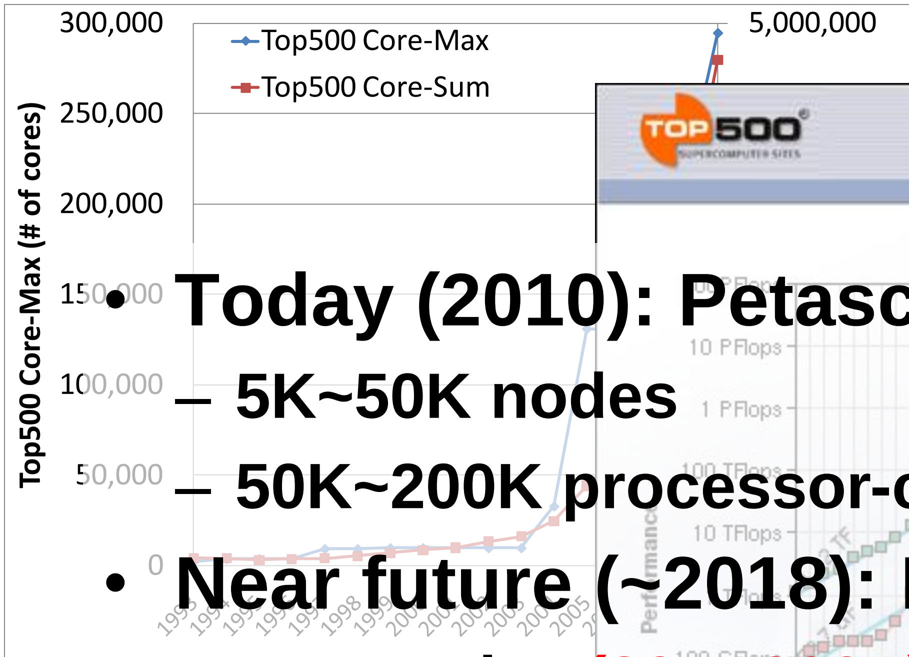
Manycore Computing

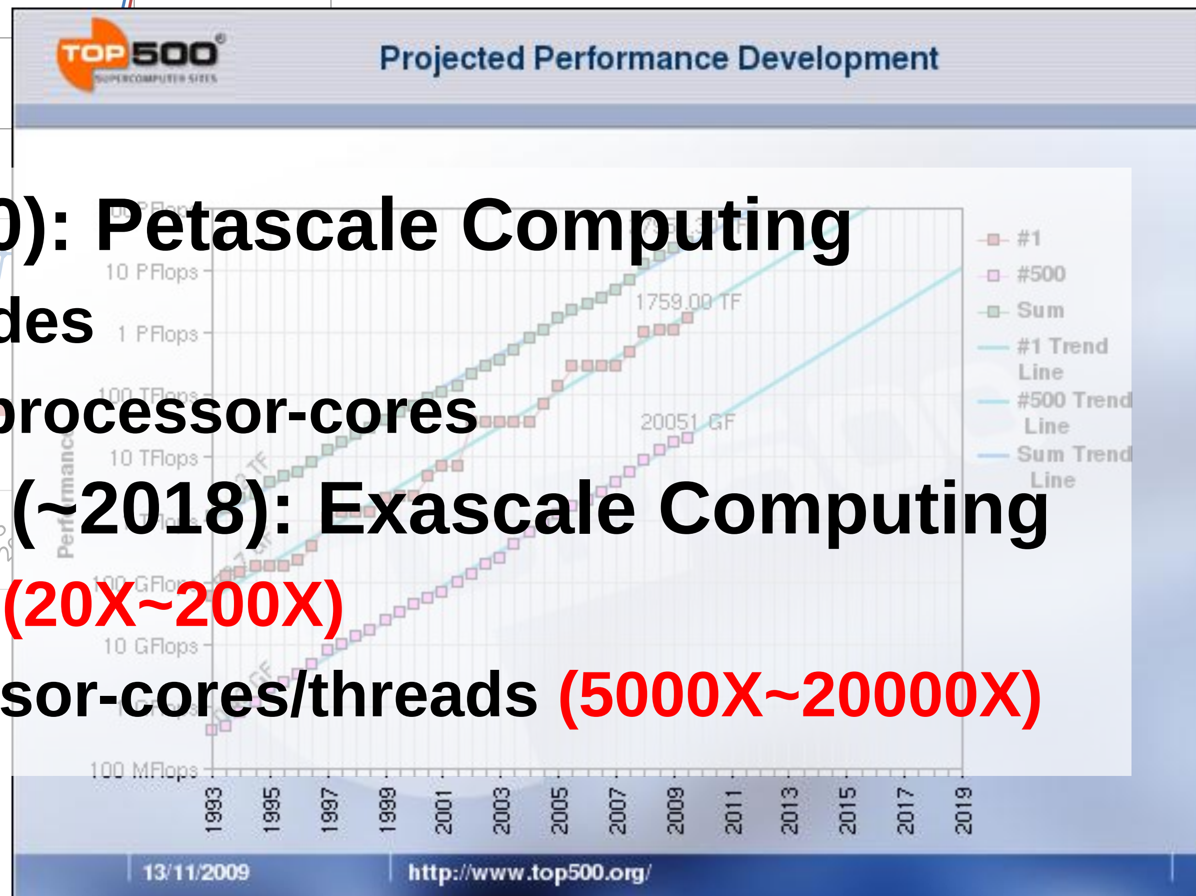


- **Today (2010): Multicore Computing**
  - **1~12 cores commodity architectures**
  - **80 cores proprietary architectures**
  - **480 GPU cores**
- **Near future (~2018): Manycore Computing**
  - **~1000 cores commodity architectures**

Pat Helland, Microsoft, The Irresistible Forces Meet the Movable Objects, November 9th, 2007

Exascale Computing





- **Today (2010): Petascale Computing**
  - **5K~50K nodes**
  - **50K~200K processor-cores**
- **Near future (~2018): Exascale Computing**
  - **~1M nodes (20X~200X)**
  - **~1B processor-cores/threads (5000X~20000X)**

Top500 Projected Development, http://www.top500.org/lists/2009/11/performance_development

Cloud Computing

- Relatively new paradigm… 3 years old
- Amazon in 2009
  - 40K servers split over 6 zones
    - 320K-cores, 320K disks
    - $100M costs + $12M/year in energy costs
    - Revenues about $250M/year
- Amazon in 2018
  - Will likely look similar to exascale computing
    - 100K~1M nodes, ~1B-cores, ~1M disks
    - $100M~$200M costs + $10M~$20M/year in energy
    - Revenues 100X~1000X of what they are today

Common Challenges

- Power efficiency
  - Will limit the number of cores on a chip (Manycore)
  - Will limit the number of nodes in cluster (Exascale and Cloud)
  - Will dictate a significant part of the cost of ownership
- Programming models/languages
  - Automatic parallelization
  - Threads, MPI, workflow systems, etc
  - Functional, imperative
  - Languages vs. Middlewares

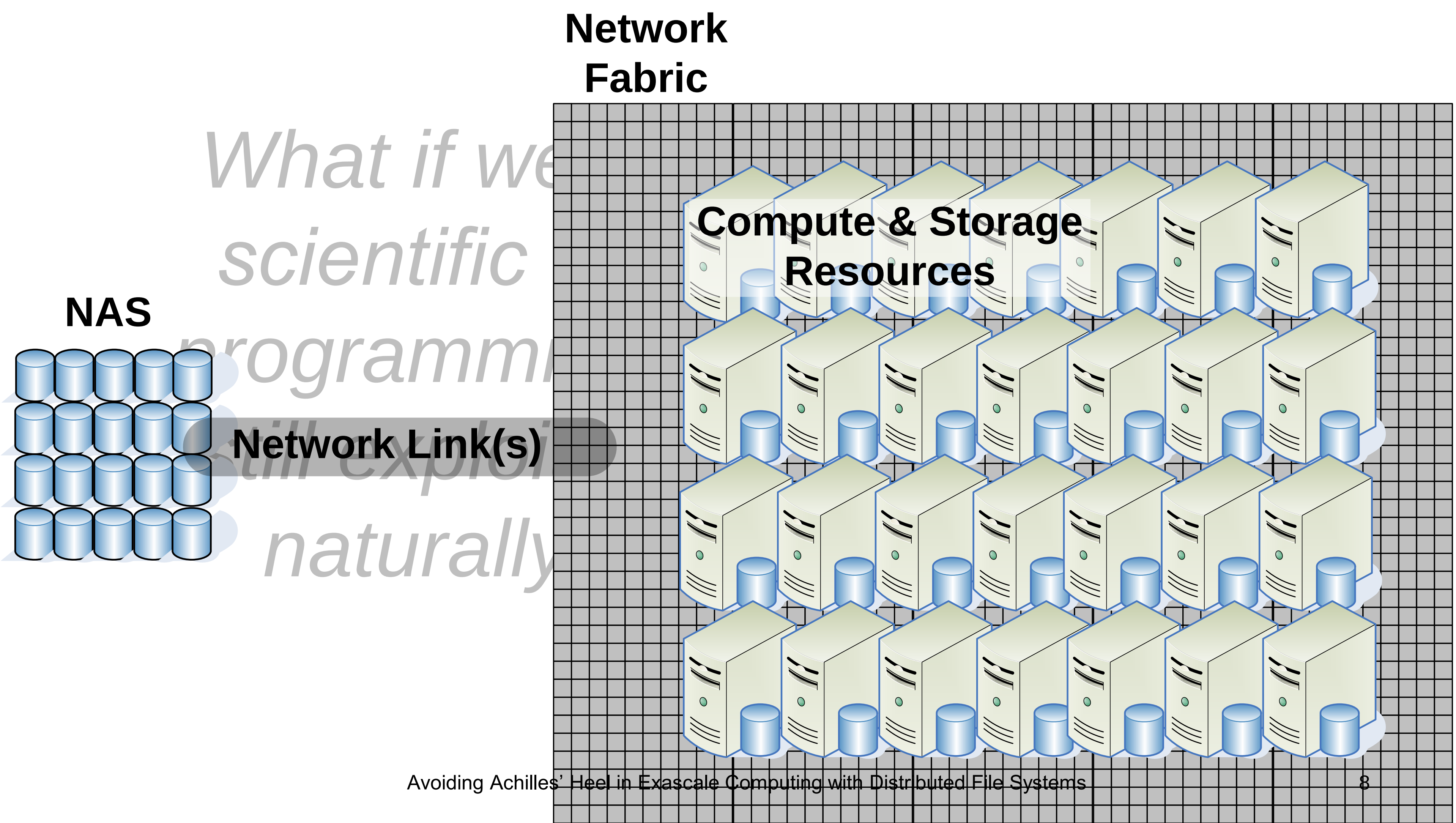
Common Challenges

- Bottlenecks in scarce resources
  - Storage (Exascale and Clouds)
  - Memory (Manycore)
- Reliability
  - How to keep systems operational in face of failures
  - Checkpointing (Exascale)
  - Node-level replication enabled by virtualization (Exascale and Clouds)
  - Hardware redundancy and hardware error correction (Manycore)

Research Directions

- ***Decentralization is critical***
  - Computational resource management (e.g. LRMs)
  - Storage systems (e.g. parallel file systems)
- ***Data locality must be maximized, while preserving I/O interfaces***
  - POSIX I/O on shared/parallel file systems ignore locality
  - Data-aware scheduling coupled with distributed file systems that expose locality is the key to scalability over the next decade

Storage System Architecture



Plan of Work

- ***Building on my own research (e.g. data-diffusion), parallel file systems (PVFS), and distributed file systems (e.g. GFS)***
- Build a distributed file system for HEC
  - It should complement parallel file systems, not replace them
- Critical issues:
  - Must mimic parallel file systems interfaces and features in order to get wide adoption
  - Must handle some workloads currently run on parallel file systems significantly better

Plan of Work (cont)

- Access Interfaces and Semantics
  - POSIX-like compliance for generality (e.g. via FUSE)
  - Relaxed semantics to increase scalability
    - Eventual consistency on data modifications
    - Write-once read-many data access patterns
- Distributed metadata management
  - Employ structured distributed hash tables like data-structures
  - Must have O(1) put/get costs
  - Can leverage network-aware topology overlays
- Distribute data across many nodes
  - Must maintain and expose data locality in access patterns

Access Patterns

- **1-many read** (all processes read the same file and are not modified)
- **many-many read/write** (each process read/write to a unique file)
- **write-once read-many** (files are not modified after it is written)
- **append-only** (files can only be modified by appending at the end of files)
- **metadata** (metadata is created, modified, and/or destroyed at a high rate).

Usage Scenarios

- **machine boot-up** (e.g. reading OS image on all nodes)
- **application loading** (e.g. reading scripts, binaries, and libraries on all nodes/processes)
- **common user data loading** (e.g. reading a common read-only database on all nodes/processes)
- **checkpointing** (e.g. writing unique files per node/process)
- **log writing** (writing unique files per node/process)
- **many-task computing** (each process reads some files, unique or shared, and each process writes unique files)

Collaborations

- Mike Wilde
  – Swift: allow Swift to scale better where parrallel file systems pose a scalability bottleneck
- Matei Ripeanu (who is also working with Mike Wilde)
  – Integrate research results into MosaStore (e.g. distributed meta-data)
- Rob Ross
  – Guidance and comparison with PVFS
- Others: Ian Foster, Kamil Iskra, Pete Beckman

More Information

- More information:
  - http://www.cs.iit.edu/~iraicu/
  - iraicu@cs.iit.edu