Exaflop Computing : Visions and Challenges: A thousand times better, not just for the High End

Peter Strazdins
Computer Systems Group,
School of Computer Science,
The Australian National University

panel discussion at the International Symposium of Parallel and Distributed Processing and Applications, Sydney, 2008

THE AUSTRALIAN NATIONAL UNIVERSITY

1 Exaflop vs Exascale Computing

- Exascale Computing: aim by 2015 to:
  - $1000\times$ RoadRunner performance: 1.6 petaflops, LINPACK (exaflops)
  - $1\times$ RoadRunner performance at $1/1000\times$ energy
  - $1000\times$ 'better' on smaller scales
- the crucial challenge, in terms of power/performance: (Peter Kogge)
  - not in the flops, but the cost to move data
  - from RAM to CPU chip, across networks
- are there other dimensions to 'better'?
  - productivity (HPCS):programming, usability (+security)
    - models: dedicated system, batch supercomputer, super-cluster / cloud
  - reliability

These only increase the challenge . . .

2 Exascale Systems

- dramatic improvements in performance / energy will hardware specialized for the key applications
  - processing: (hetero-) multicore, GPU, FGPA
  - FGPA and standard network processors / interfaces
    - the QPACE QCD architecture (Goldrian et al, 2008)
- hence heterogeneity will also need to be handled
- need hierarchy (architecture, run-time environment, algorithms) to minimize long-range data transfers
- need also programming paradigms that can handle these
  - a forerunner: the Liquid Metal project (Huang, Horman, Bacon and Rabbath, 2006)
- challenge: can all this be done with the right level of abstraction (performance and robustness vs. productivity)

3 Virtualization in Exascale Computing

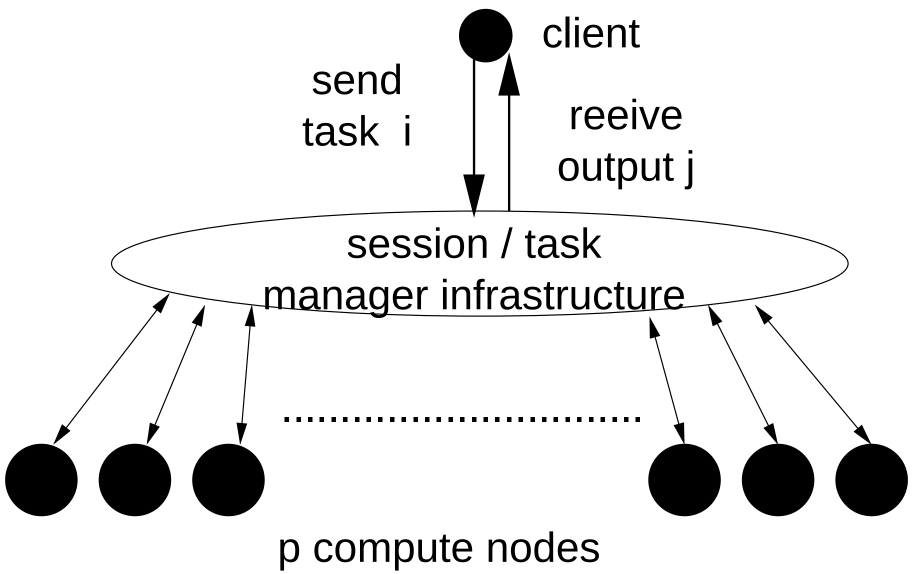
- virtualization allows users to customize their environments
- also allows for easy migration across super-cluster / cloud
  - move parallel job to more suitable parts of heterogeneous system as they become available
  - move job closer to its (possibly varying) data sources
- interesting forerunner: Snowflock Project (U. Toronto, 2008)
  - 'fork' a Xen VM to achieve parallelism; surprisingly low overhead
  - current motivation: cloud computing (biosciences)
- extra performance overheads still to be addressed (e.g. heterogeneous multicore)

4 Programming Paradigms: Productivity and Reliability

- MPI has been long denounced wrt. productivity (e.g. Thomas Sterling, Cluster'04)
  - worst still under the assumption of heterogeneity
- alternatives such as cluster-aware OpenMP have difficulties achieving comparable (even satisfactory) performance
- some reasons (page-based software DSM systems):
  - granularity of communications; pages often too small
  - difficulty to fetch in advance, use collective communications
  - simply parallelizing loops in existing algorithms may be sub-optimal
- neither model can easily be made reliable against node failures
  - (distributed) processor state is hard to save
  - typically alleviated by explicit checkpointing
- do the HPCS languages address this any better?

5 Lessons from the Grid: Fault-Tolerance, Heterogeneity-oblivious, Programmability

- frameworks for large sets of embarrassingly parallel tasks
  - e.g. Symphony middleware (for 'coarse' tasks on enterprise grids) (Platform (TM) Computing)



- task i: receives input, performs local computation, sends output
- naturally load balancing if number of tasks $>> p$
- fault-tolerance: if a compute node fails, simply restart its task

- can we extend this model for general parallel computation?
  - with use of a smart FT 'fabric' to hold/cache/move global data