Fault Tolerance For Sparse Linear Algebra Computations Implemented In A Grid Environment

Experiments with Fault Tolerant Linear Algebra Algorithms

Jack Dongarra
Jeffery Chen
Zhiao Shi
Asim YarKhan

University of Tennessee

VGrADS
Virtual Grid Application Development Software Project

Fault Tolerance: Motivation

- Interested in using the VGrADS framework to find resources to solve problems and increase the ease of use in a fault prone system.
  - Application driven adaptation
- With Grids and some parallel systems there's an increased probability of a system or network failure
  - Mean Time to Failure is growing shorter as system's size increase.
- By monitoring, one can identify
  - Performance problems
  - Failure probability
    - Fault prediction
    - Migration opportunities.
  - Prepare for fault recovery

- Large-scale fault tolerance
  - Self adaptation: resilience and recovery
  - Predictive techniques for probability of failure
    - Resource classes and capabilities
    - *Coupled to application usage modes*
  - Resilience implementation mechanisms
    - Adaptive checkpoint frequency
    - In memory checkpoints

VGrADS
Virtual Grid Application Development Software Project

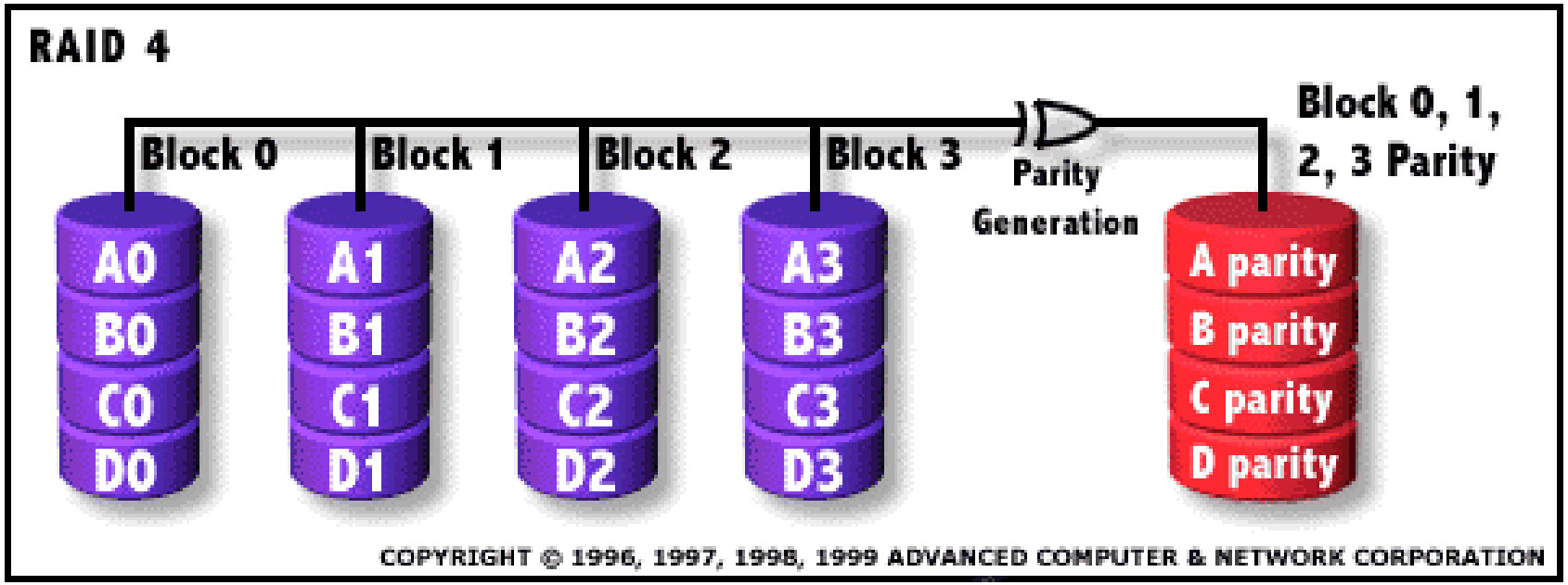
Fault Tolerance - Diskless Checkpointing Built into Software

- Checkpointing to disk is slow.
  - May not have any disks on the system.
- Have extra checkpointing processors allocated.
- Use "RAID like" checkpointing to processor.
- Maintain a system checkpoint in memory.
  - All processors may be rolled back if necessary.
  - Use k extra processors to encode checkpoints so that if up to k processors fail, their checkpoints may be restored (Reed-Solomon encoding).
- Idea to build into library routines.
  - We are developing this for iterative solvers, *Ax=b*.
  - Not transparent, has to be built into the algorithm.
- Use VGrADS virtualization to hide complexity

VGrADS
Virtual Grid Application Development Software Project

How Raid for a Disk System Works

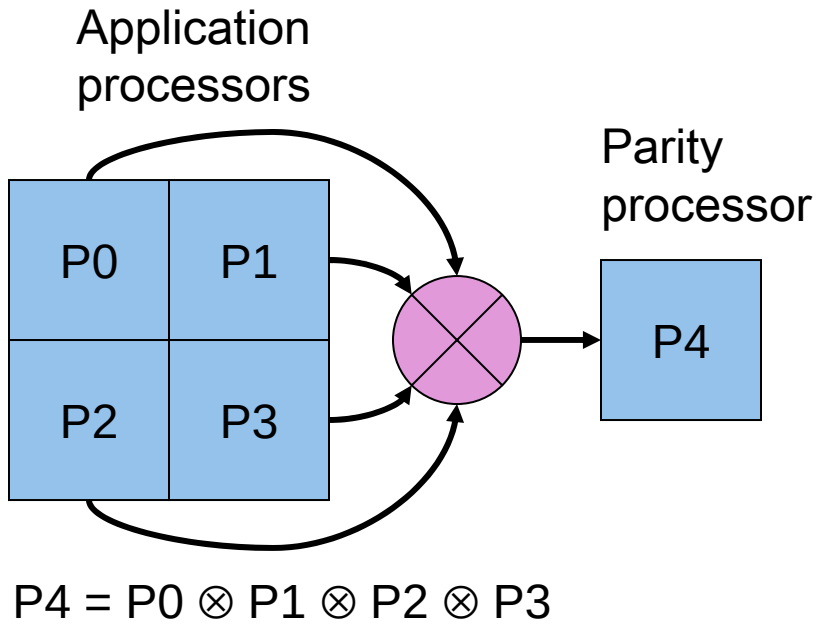
- **Similar to RAID for disks.**



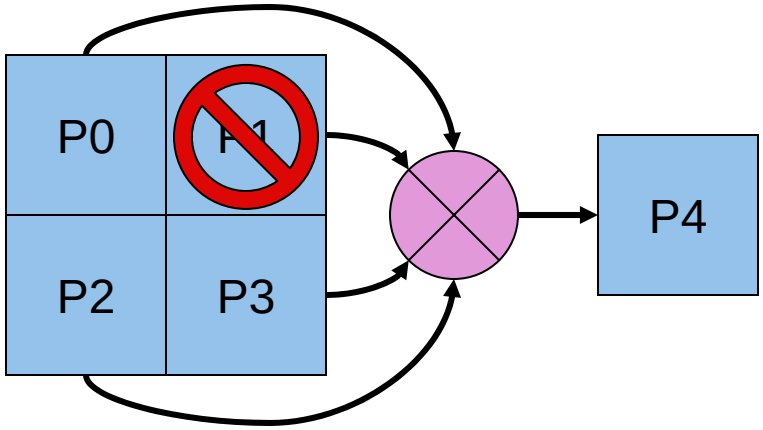
- **If X = A XOR B then this is true:**
  - X XOR B = A
  - A XOR X = B

Diskless Checkpointing

- The N application processors (4 in this case) each maintain their own checkpoints locally.
- K extra processors maintain coding information so that if 1 or more processors fail, they can be replaced.
- Here described for k=1 (parity).
- If a single processor fails, then its state may be restored from the remaining live processors.

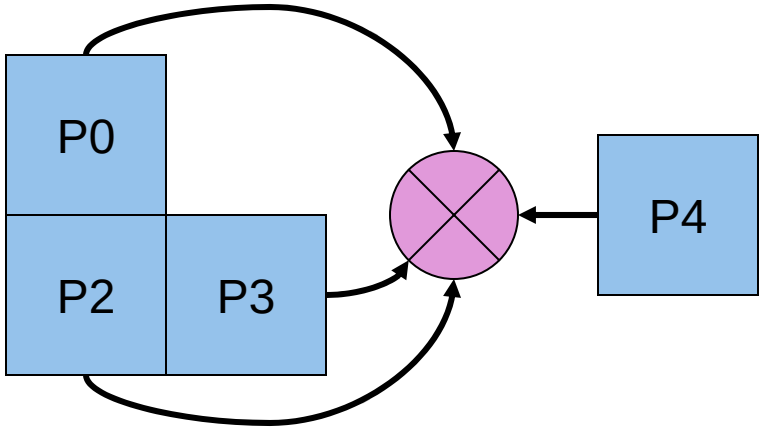


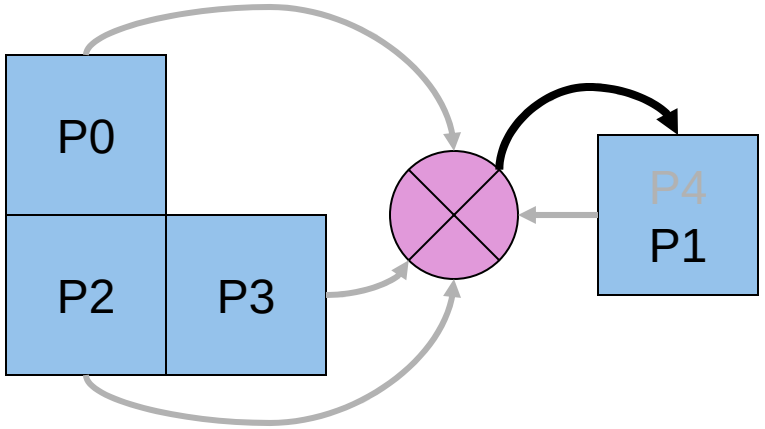
Diskless Checkpointing



- When failure occurs:
  - Control passes to user supplied handler
  - "XOR" performed to recover missing data
  - P4 takes on role of P1
  - Execution continue

P4 takes on the identity of P1
and the computation continues.



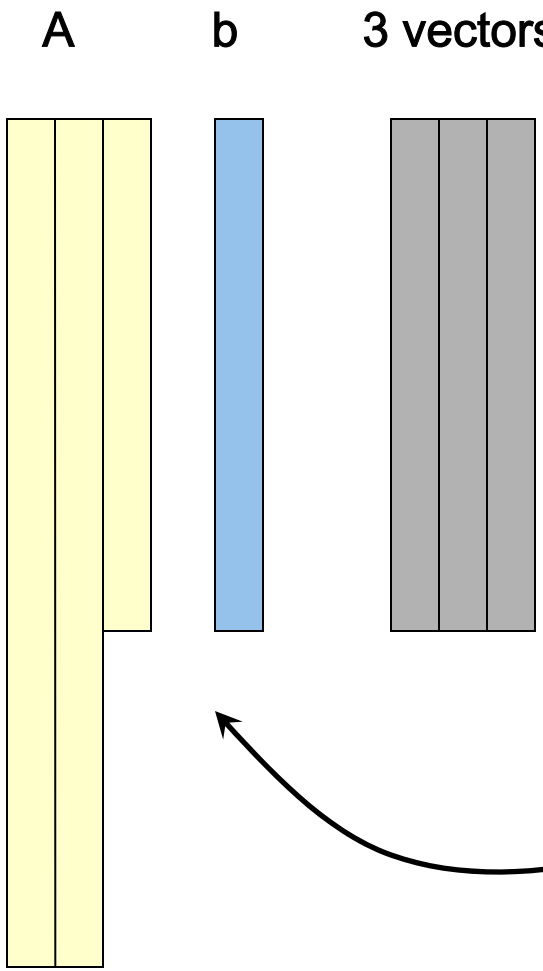


A Fault-Tolerant Parallel CG Solver

- Tightly coupled computation.
  - Not expecting to do wide area distributed computing.
  - Cluster based is ideal.
  - Issues on how many processors and checkpoint processors "optimal" for given problem, including failure scenario. May vary from run to run.
- Do a "backup" (checkpoint) every *j* iterations for changing data.
  - Requires each process to keep copy of iteration changing data from checkpoint.
- First example can survive the failure of a single process.
- Dedicate an additional process for holding data, which can be used during the recovery operation.
- For surviving *k* process failures ($k << p$) you need *k* additional processes (second example).

CG Data Storage

Think of the data like this



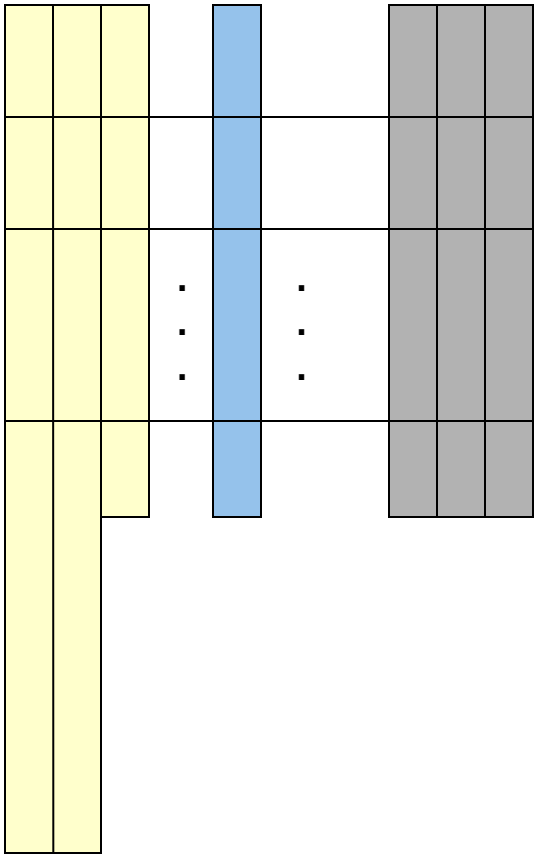
3 vectors change every iteration

Checkpoint A and b
Initially, data is fixed throughout the iteration

Parallel Version

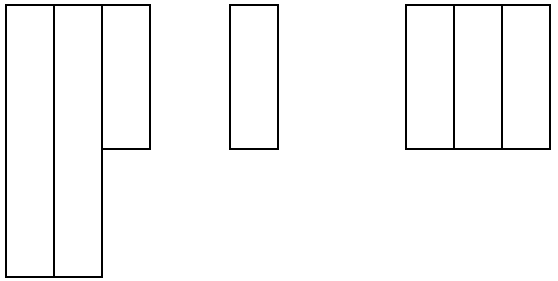
Think of the data like this

A b 3 vectors



Think of the data like this on each processor

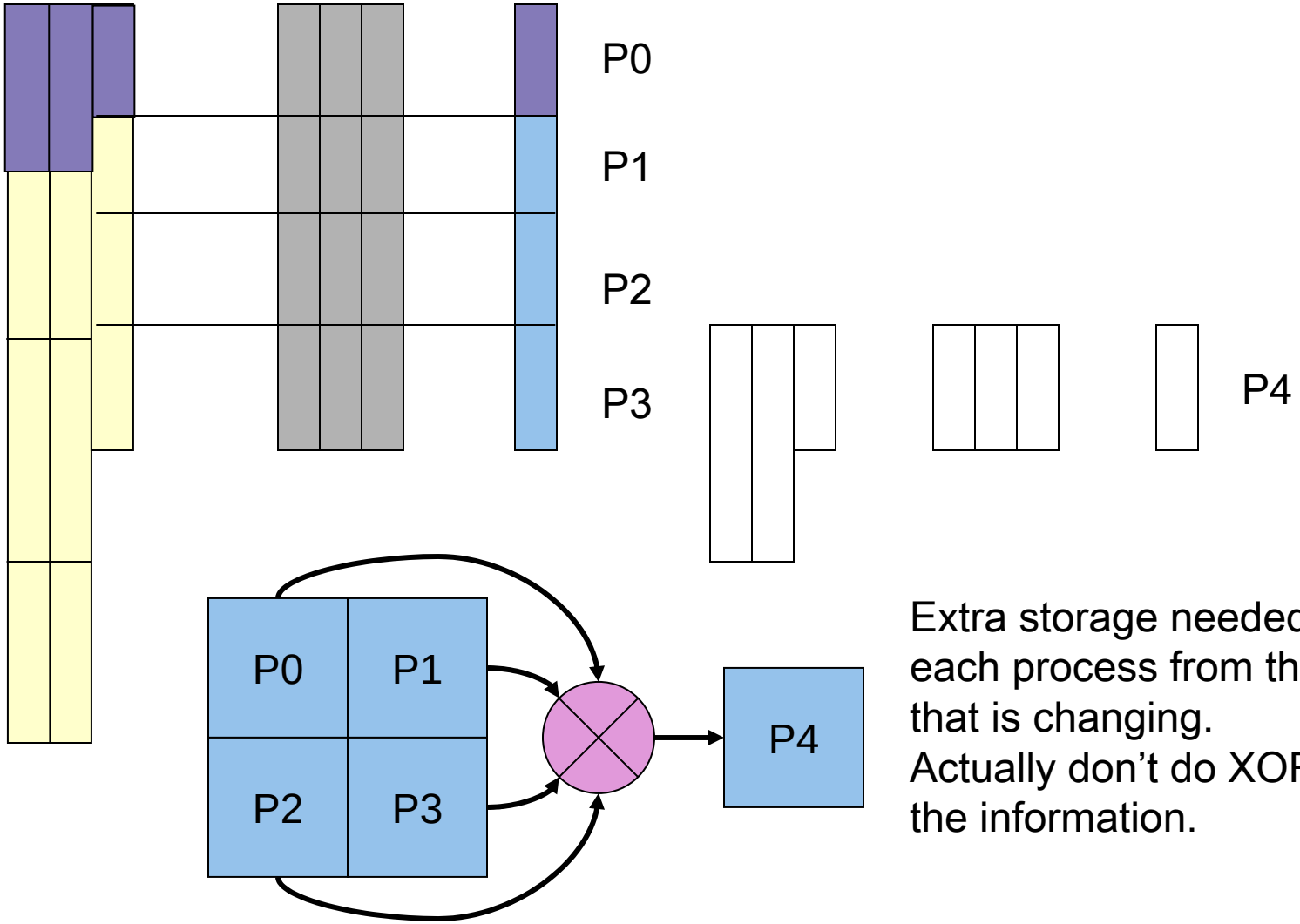
A b 3 vectors



No need to checkpoint each iteration, say every *j* iterations.

Need a copy of the 3 vectors from checkpt in each processor.

Diskless Version



Extra storage needed on each process from the data that is changing.
Actually don't do XOR, add the information.

FT PCG Algorithm Analysis

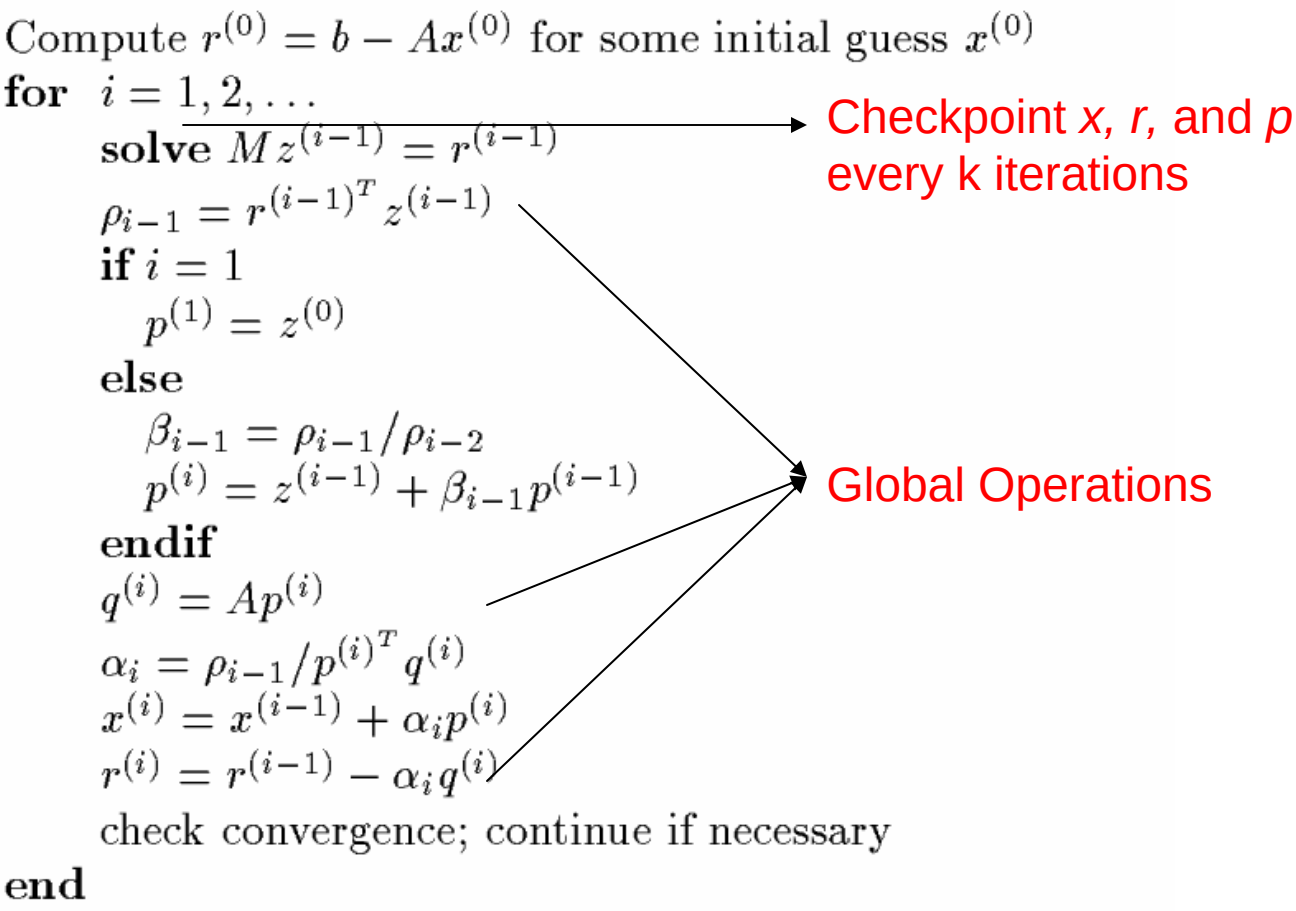
Compute $r^{(0)} = b - Ax^{(0)}$ for some initial guess $x^{(0)}$
for $i = 1, 2, \ldots$
  solve $Mz^{(i-1)} = r^{(i-1)}$
  $\rho_{i-1} = r^{(i-1)^T} z^{(i-1)}$
  if $i = 1$
    $p^{(1)} = z^{(0)}$
  else
    $\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$
    $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$
  endif
  $q^{(i)} = Ap^{(i)}$
  $\alpha_i = \rho_{i-1}/p^{(i)^T} q^{(i)}$
  $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$
  check convergence; continue if necessary
end

Global Operations

Global operation in PCG: three dot product, one preconditioning, and one matrix vector multiplication.

Global operation in Checkpoint: encoding the local checkpoint.

FT PCG Algorithm Analysis



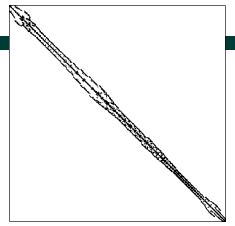
Global operation in PCG: three dot product, one preconditioning, and one matrix vector multiplication.

Global operation in Checkpoint: encoding the local checkpoint.

Global operation in checkpoint can be localized by sub-group.

VGrADS
Virtual Grid Application Development Software Project

PCG: Performance with Different MPI Implementations



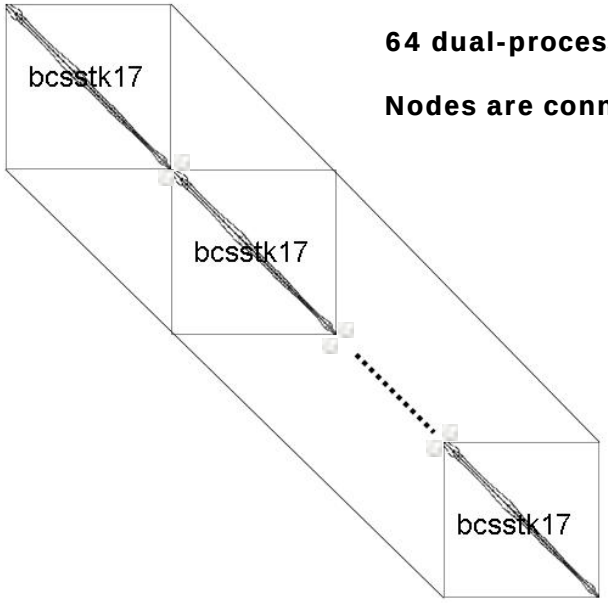
bcsstk17:

The size is:
10974 x 10974

Non-zeros:
428650

Sparsity:
39 non-zeros per row on average

Source:
Linear equation from elevated pressure vessel



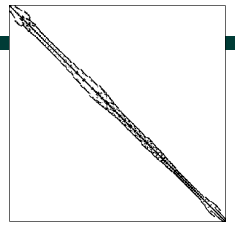
64 dual-processor 2.4 GHz AMD Opteron nodes

Nodes are connected with a Gigabit Ethernet.

N	Procs	LAM-7.0.4	MPICH2-1.0	FT-MPI
165K	15	522.5	536.3	517.8
329K	30	532.9	542.9	532.2
658K	60	545.5	553.0	546.5
1317K	120	674.3	624.4	622.9

http://icl.cs.utk.edu/ft-mpi/

PCG: Performance with Different MPI Implementations



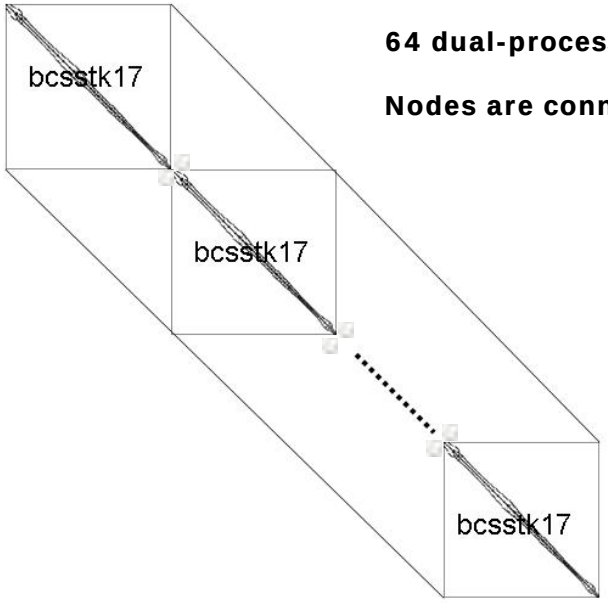
bcsstk17:

The size is:
10974 x 10974

Non-zeros:
428650

Sparsity:
39 non-zeros per row on average

Source:
Linear equation from elevated pressure vessel



64 dual-processor 2.4 GHz AMD Opteron nodes

Nodes are connected with a Gigabit Ethernet.

N	Procs	LAM-7.0.4	MPICH2-1.0	FT-MPI	FT-MPI ckpt /2000 iters	FT-MPI exit 1 proc @10000 iters
165K	15	522.5	536.3	517.8	518.9	521.7
329K	30	532.9	542.9	532.2	533.3	537.5
658K	60	545.5	553.0	546.5	547.8	554.2
1317K	120	674.3	624.4	622.9	624.4	637.1

http://icl.cs.utk.edu/ft-mpi/

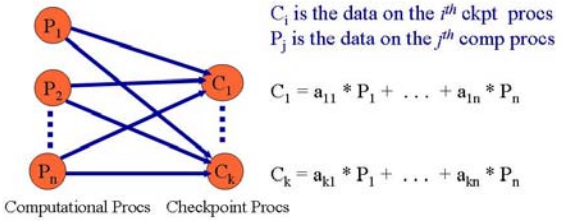
VGrADS
Virtual Grid Application Development Software Project

Protecting for More Than One Failure: Reed-Solomon (Checkpoint Encoding Matrices)

- In order to be able to recover from any k (≤ number of checkpoint processes) failures, need a checkpoint encoding.
- With one checkpoint process we had:
  - —P sets of data and a function A such that
  - —C=A*P where $P=(P_1,P_2,...P_p)^T$;
    - C: Checkpoint data (C and P_i same size)
    - With A = (1, 1, ..., 1)
    - $C = a_1P_1 + a_2P_2 + ...+ a_p P_p$; C = A*P
    - To recover P_k;

      solve $P_k = (C-a_1P_1-a_{k-1}P_{k-1}-a_{k+1}P_{k+1}-a_pP_p)/a_k$

Could use GF(2). Signal processing aps do this. In that case, A is Vandermonde or Cauchy matrix. (Need to have any subset of A be non singular.)
We use A as a random matrix.

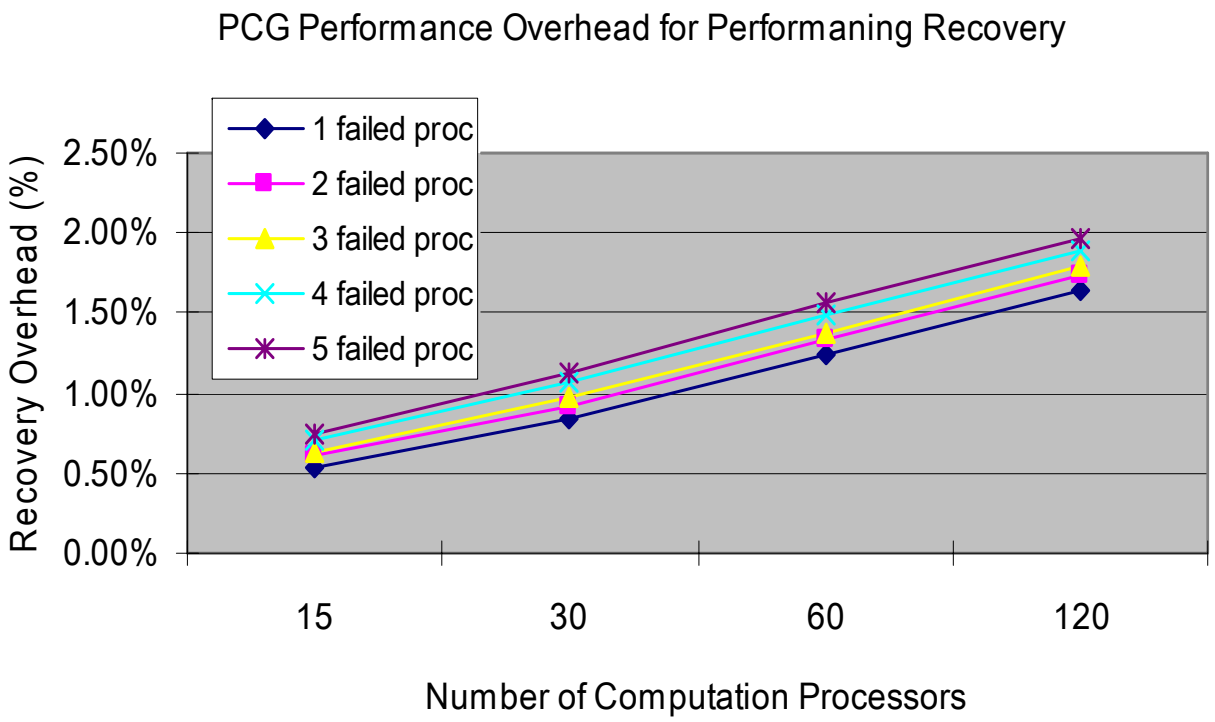


- With k checkpoints we need a function A such that

  C=A*P where $P=(P_1,P_2,...P_p)^T$;
  - C: Checkpoint data $C = (C_1,C_2,...C_k)^T$ (C_i and P_i same size).
  - A: Checkpoint-Encoding matrix A is k x p (k << p):

    A= k x p
- When h failures occur, recover the data by taking the h x h submatrix of A, call it A', corresponding to the failed processes and solving A'P' = C'; to recover the h "lost" P's.
  - —A' is the h x h submatrix.
  - —C' is made up of the surviving h checkpoints.

PCG: Performance Overhead of Performing Recovery

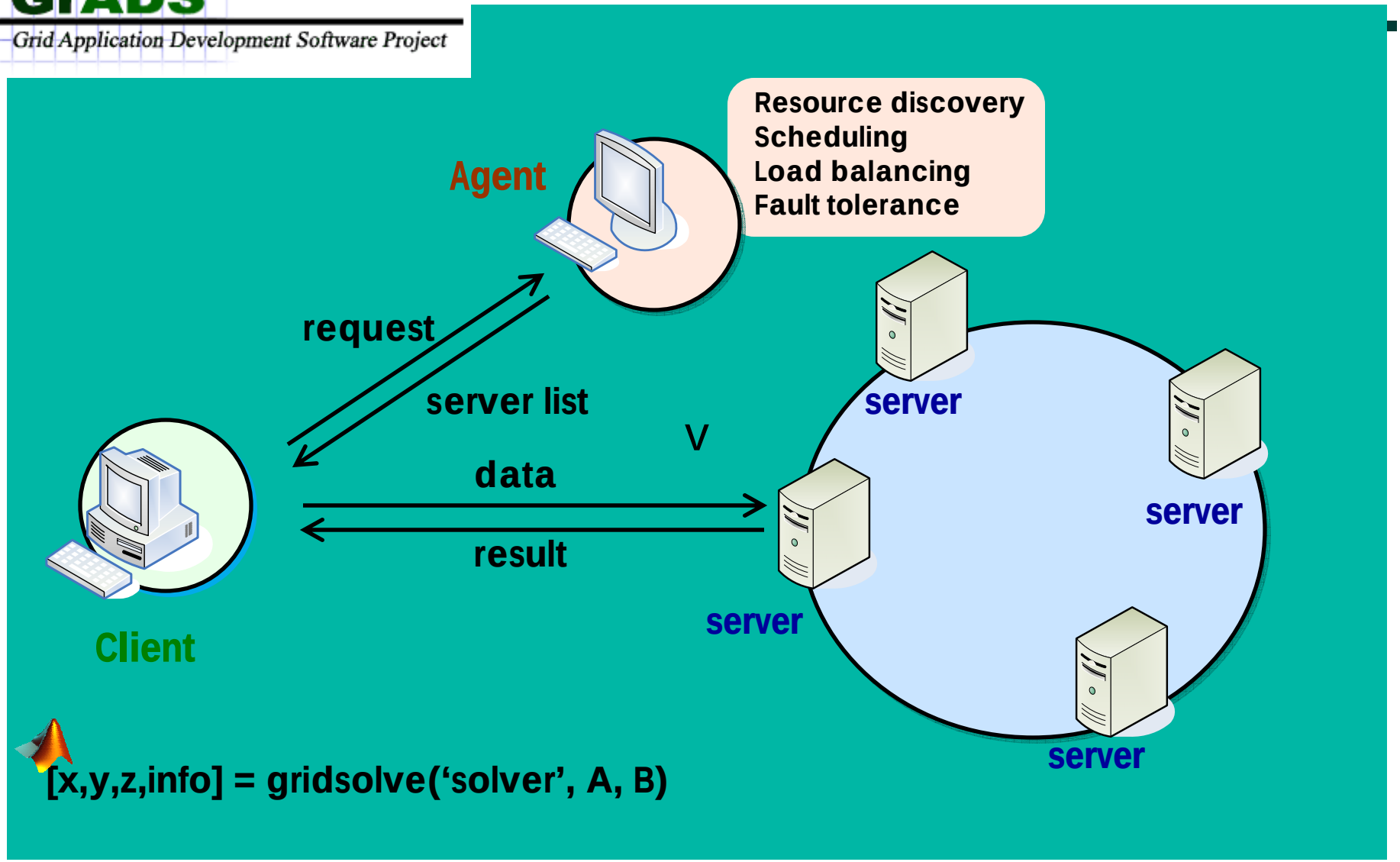


T (ckpt T)	0 proc	1 proc	2 proc	3 proc	4 proc	5 proc
15 comp	517.8	521.7 (2.8)	522.1 (3.2)	522.8 (3.3)	522.9 (3.7)	523.1 (3.9)
30 comp	532.2	537.5 (4.5)	537.7 (4.9)	538.1 (5.3)	538.5 (5.7)	538.6 (6.1)
60 comp	546.5	554.2 (6.9)	554.8 (7.4)	555.2 (7.6)	555.7 (8.2)	556.1 (8.7)
120 comp	622.9	637.1 (10.5)	637.2 (11.1)	637.7 (11.5)	638.0 (12.0)	638.5 (12.5)

NSF MIDDLEWARE INITIATIVE

GrADS
Grid Application Development Software Project

GridSolve Architecture

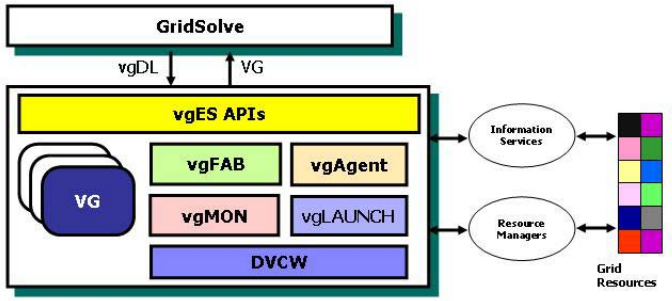


VGrADS
Virtual Grid Application Development Software Project

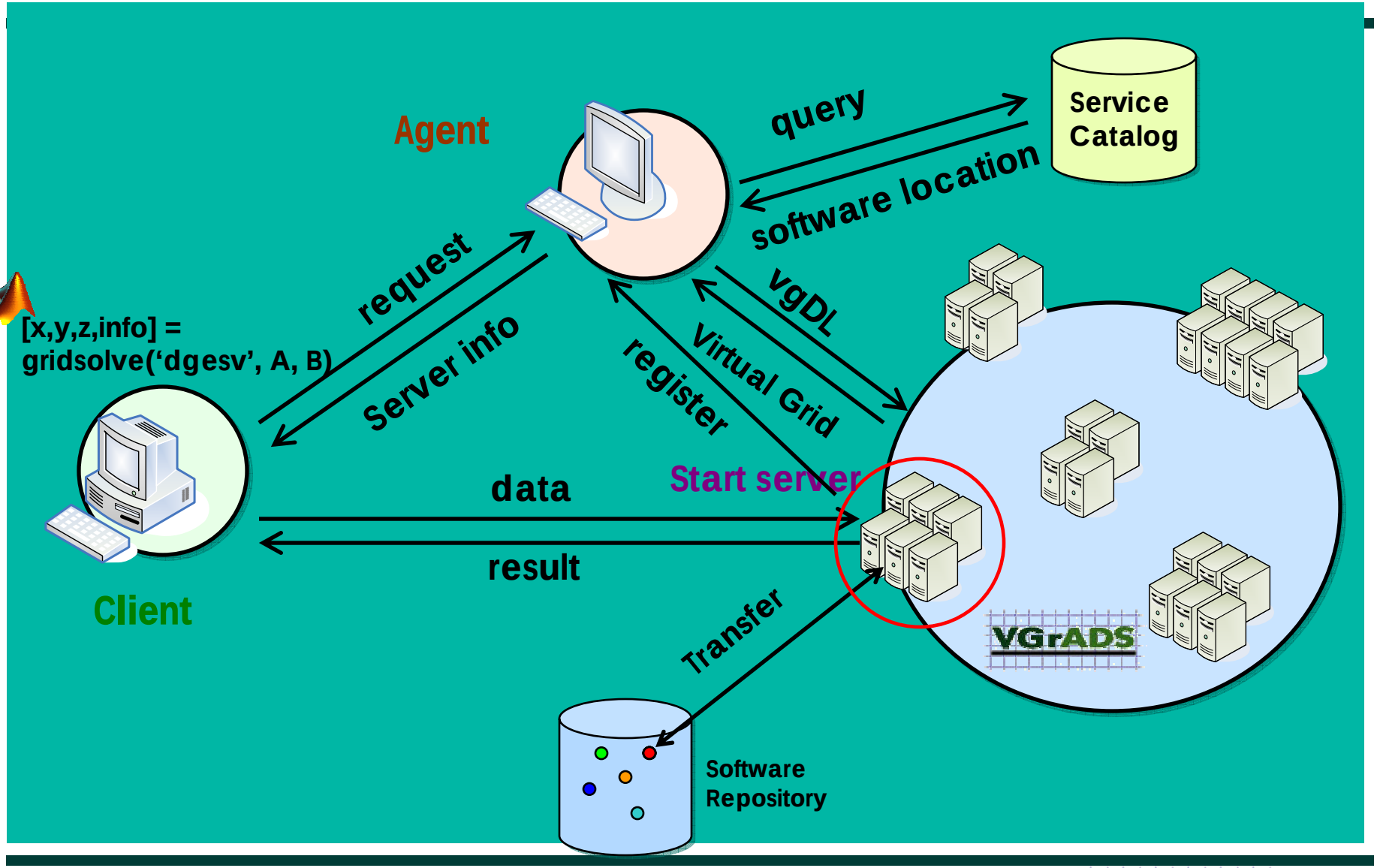
GridSolve Usage with VGrADS

- Simple-to-use access to complicated software libraries
- Selection of best machine in your grid to service user request
- Portability
  - Non-portable code can be run from a client on an architecture as long as there is a server provisioned with the code
- Legacy codes easily wrapped into services

- Plug into VGrADS Framework
- Using the vgES for resource selection and launching of application:
  - Integrated performance information
  - Integrated monitoring
  - Fault prediction
  - Integrating the software and resource information repositories



VGrADS/GridSolve Architecture



VGrADS
Virtual Grid Application Development Software Project

Agent

- Agent is specific for the client
  - —Initially agent contains no resource information; obtained from vgES
- Agent requests information from the service catalog about the possible services and their complexity in order to estimate the resources required (vgDL)
- For each service request
  - —Estimates resources required
    - vgDL spec: vgdl = Clusterof<node>[N]; node = {node.memory > 500MB, node.speed > 2000};
    - vgid = vgCreateVG(vgserver, vgdl, 1000, ns-server-script)
  - —Return the set of resources to the client
  - —The ns-server-script fetches and deploys needed services on selected VGrADS resources

Next Steps

- Software to determine the checkpointing interval and number of checkpoint processors from the machine characteristics.
  - Perhaps use historical information.
  - Monitoring
  - Migration of task if potential problem
- Local checkpoint and restart algorithm.
  - Coordination of local checkpoints.
  - Processors hold backups of neighbors.
- Have the checkpoint processes participate in the computation and do data rearrangement when a failure occurs.
  - Use p processors for the computation and have k of them hold checkpoint.
- Generalize the ideas to provide a library of routines to do the diskless check pointing.
- Look at "real applications" and investigate "Lossy" algorithms.

Additional Details and Related Posters

- VGrADS and GridSolve
  - Zhiao Shi, UTK
- Optimal Checkpoint Scheduling
  - Dan Nurmi, UCSB
- Scheduling Compute Intensive Apps in Volatile Env.
  - Richard Huang, UCSD
- Adaptive Resource Environments for HPG Apps
  - Jerry Chou, UCSD

Publications:

- *Condition Numbers of Gaussian Random Matrices,* Zizhong Chen and Jack Dongarra, to appear SIAM Matrix Analysis and Applications.
- *Building Fault Survivable MPI Programs with FTMPI Using Diskless Checkpointing*, Zizhong Chen, Graham E. Fagg, Edgar Gabriel, Julien Langou, Thara Angskun, George Bosilca, and Jack Dongarra, accepted PPoPP 2005.