Failure-Aware Checkpointing in Fine-Grained Cycle Sharing Systems

Xiaojuan Ren, Rudolf Eigenmann, Saurabh Bagchi
School of ECE, Purdue University

Background on Fine-Grained Cycle Sharing Systems

- Cycle sharing: SETI@Home, Entropia
  - Harvest idle cycles of Internet connected PCs
    - Desktop machines: busy in the day, idle at night
    - CPU cycles are perishable: wasted if not used
  - Preserve PC owners' priority in resource sharing
    - Resource becomes unavailable if owners are *active*
- Fine-grained cycle sharing
  - Allows guest jobs to coexist on a machine with host ("submitted by owner") jobs
  - Resource becomes unavailable if slowdown of host jobs is observable

Fine-Grained Cycle Sharing: FGCS

Trouble in “FGCS Land”

- Uncertainty of execution environment with fluctuating resource availability
  - Resource contention from guest jobs
  - Resource revocation by machine owner
  - Software-hardware faults
  - Abrupt removal of machine from network
- Resource unavailability is not rare
  - About 400 occurrences in traces collected during 3 months on a typical host machine

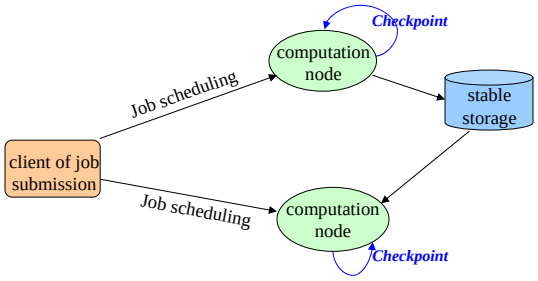
How to Handle Fluctuating Resource Availability?

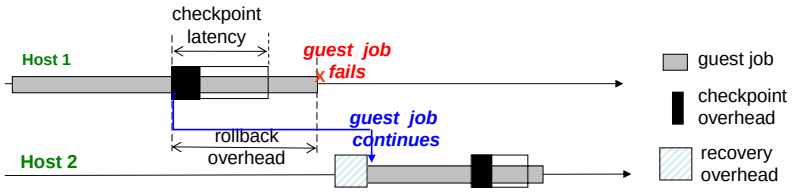
- Reactive approach
  - Does not have knowledge of future failures
  - Does nothing until failures happen: restart job on a different machine
- Proactive resource allocation
  - Predicts when resource will become unavailable
  - Schedules a job to the machine with high availability
  - Applies predictive knowledge in managing job execution

We have worked on…

- Resource availability prediction *(HPDC'06)*
  - *TR*: temporal reliability
  - Algorithm: Semi-Markov Process models
  - Accuracy: > 86.5% on average
- Proactive job scheduling *(JoGC'07)*
  - Two-phase algorithm
  - Outperform reactive algorithms: improve job *makespan* by 14% on average
  - Long jobs are still failure-prone (> 5 hours)

Checkpointing and Rollback





Apply Checkpointing in FGCS?

- Lack of dedicated storage
  - Storage hosts: hosts that contribute disk space to guest users
  - Fluctuating availability
- Load of transferring checkpoints
  - Network links shared between hosts and guests
  - Checkpoint transfers over the Internet
- Checkpoint intervals
  - Checkpoint overhead v.s. Rollback overhead

Contributions on Failure-Aware Checkpointing

- Select reliable and efficient storage out of failure-prone hosts
  - Criteria of selection:
    - Re-execution cost due to a lost checkpoint
    - Network performance of transferring checkpoints
  - To enhance fault-tolerance: encode a checkpoint into redundant fragments
- Determine checkpoint intervals
  - Expected rollback overhead is a function of resource availability
  - Checkpoint only if the expected rollback overhead higher than checkpoint overhead

Related Work

- Production Systems
  - Dedicated checkpoint servers
  - Pre-defined checkpoint periodicity
  - Scheduling of checkpoint transfers
- Research Efforts
  - Failure awareness in checkpointing/migration
  - Analytical methods to compute checkpoint intervals
  - Fault-tolerant storage: erasure codes

Framework of Checkpointing

- Create checkpoints
  - Condor's standalone checkpoint library
- Encode checkpoints
  - Information dispersal algorithm (IDA)
    - A checkpoint of size *n* → *m+k* fragments of size *n/m*
    - Any *m* fragments → the original checkpoint
  - Overhead: < 20 seconds for checkpoint size of 200 MB
- Store checkpoints
  - Save to *m+k* storage hosts – checkpoint repository
  - Overlap network transfer with job execution
- Recover from failures
  - Select a new computation host
  - Collect *m* checkpoint fragments
  - Restart from the checkpointed state

Checkpoint Repository Selection

- Problem description
  - select *m+k* checkpoint repositories from *V* storage hosts
- Optimization objectives
  - Network performance (*N*)
    - How long a checkpoint uses the network – latency
    - End-to-end effective network bandwidth $\frac{n}{m}/bw$
  - Expected re-execution cost (*R*)
    - Storage availability -- *TRS*
    - More than k repositories fail at the time of job recovery $1-\prod_{i=1}^{V} TRS'_i(t)$

Two Optimization Schemes

- Optimistic
  - Assume relatively static storage availability
  - The selected repositories for a specific computation host are used during the entire execution on this host
  - Expected recovery (failure) time: Mean Time To Failure of the computation host, $MTTF_{cmp}$ $\int_0^{\infty} TR(t)\,dt$
- Pessimistic
  - Storage availability fluctuates highly
  - Select repositories before each checkpoint commit
  - Probability of the computation host failing during the next checkpoint interval (*CI*) while already surviving to the current time, *t* $\frac{1-TR(t+CI)}{TR(t)}$

0/1 Programming Models

Optimistic scheme

$$\min\{\frac{MTTF_{cmp}}{CI} * \sum_{i=1}^{V}(Ci * \frac{n}{m} / bw_i) - (T_{curr} + MTTF_{cmp}) * \prod_{i=1}^{V} TRS'(MTTF_{cmp})\}$$

$$\sum_{i=1}^{V} C_i = m + k$$

$$TRS'(t) = \max[1 - C_i, TRS_i(t)]$$

Pessimistic scheme

$$\min\{\sum_{i=1}^{V}(Ci * \frac{n}{m} / bw_i) - T_{curr} * \prod_{i=1}^{V} TRS'(CI) * \frac{1 - TR(t + CI)}{TR(t)}\}$$

$$\sum_{i=1}^{V} C_i = m + k$$

$$TRS'(t) = \max[1 - C_i, TRS_i(t)]$$

T_{curr} – the time units spent on performing useful computation for the guest job so far;

C_i -- is 1 if storage host *i* is selected, otherwise it is 0

Greedy Algorithm to Solve the Models

- Initialization
  - Compute TRS_i and N_i ($\frac{n}{m} / bw_i$)
  - Objective function is 0
- Bootstrap
  - Rank the storage hosts by N_i (in increasing order) and TRS_i (in decreasing order), respectively
  - Select the hosts appearing in the first $m+k$ elements of both rankings
  - Update the objective function
- Iteration
  - Update the objective function by including one unselected host at a time
  - Select the host causing the minimum increase to the objective function
  - If #(selected hosts) = $m+k$, terminate

Determine Checkpoint Intervals

- One-step look-ahead heuristic
  - Compare two costs at each *step*
    - Checkpointing immediately: checkpoint overhead + rollback overhead
    - Postponing checkpointing to the next step: rollback overhead

$$C + \int_t^{t+\delta t} (s-t)\frac{f(s)}{TR(t)}ds \le \int_t^{t+\delta t} (s-t_0)\frac{f(s)}{TR(t)}ds$$

$$C \le (t-t_0) * \frac{TR(t) - TR(t+\delta t)}{TR(t)}$$

t_0 is the last time of checkpointing

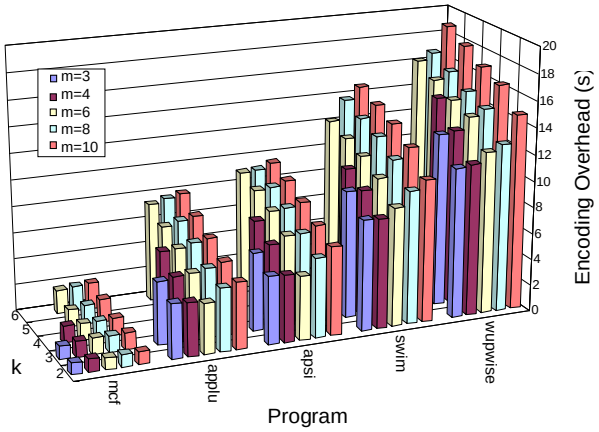
- Step length
  - Not too small
    - Checkpoint interval > checkpoint latency
  - Not too large
    - Undesirable rollback overhead

Experiments

- Performance of checkpointing implementation
  - Checkpoint overhead: checkpoint creation + IDA encoding
  - Recovery overhead: network transfer + IDA decoding
- Failure-aware checkpointing
  - Metric: average job makespan
  - Trace-based simulation
    - Traces: 1800 machine-day traces, Sep. 2006 – Nov. 2006
    - Simulator: GridSim (http://gridbus.csse.unimelb.edu.au/gridsim/)
  - Checkpoint repository selection
    - Compare with random selection
  - Checkpoint intervals
    - Compare with checkpointing at fixed periodicities
  - Failure-aware checkpointing
    - Compare with Condor: fixed checkpoint periodicity + dedicated checkpoint server

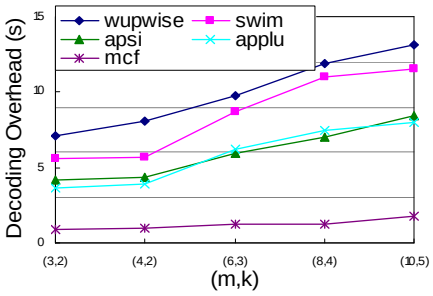
SPEC CPU 2K benchmark
Pentium IV 2.0 GHz machine with an ATA-100 (7200 RPM) hard drive

Program	mcf	applu	apsi	swim	wupwise
Checkpoint size (MB)	22.5	105.5	120.6	154.1	175.4
Overhead of generating a checkpoint to the local disk (s)	13.72	45.14	53.32	58.29	67.71

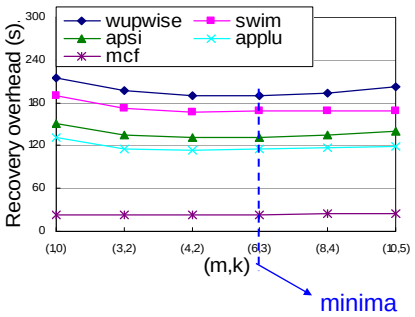


Recovery Overhead

Overhead of IDA decoding



Recovery overhead



Computation host: Pentium IV 2.0 GHz machine
Storage hosts: Pentium IV 1.5 GHz machines
10 Mbps network links

Parameter Settings for Simulation

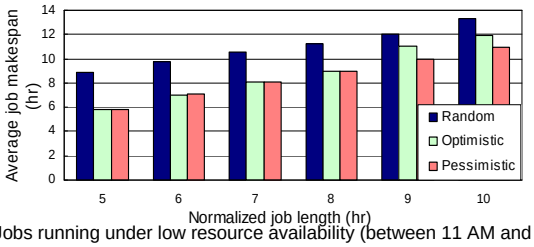
Parameters of Checkpointing

Checkpoint size, *n*	(100, 500) MB
IDA	*m* = 6, *k* = 3
Checkpoint overhead (s)	0.43*n + 5.63
Decoding overhead (s)	0.06*n – 0.29

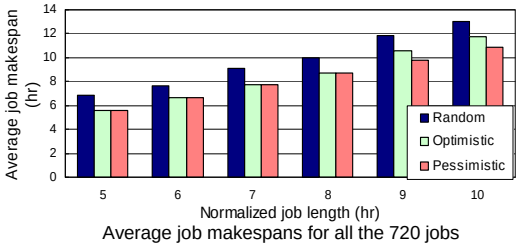
- Traces of host workload and availability
  - A storage host is available about 70% of the time on average
- Simulated testbed
  - 20 computation hosts: Pentium IV 2.0 GHz machine
  - 20 storage hosts: Pentium IV 1.5 GHz machine
  - 10 Mbps network links
- Guest jobs
  - 720 jobs with submission time between 12 AM and 10 PM
  - Normalized job length between 5 -- 10 hours

Checkpoint Repository Selection

checkpoint interval = 1hour



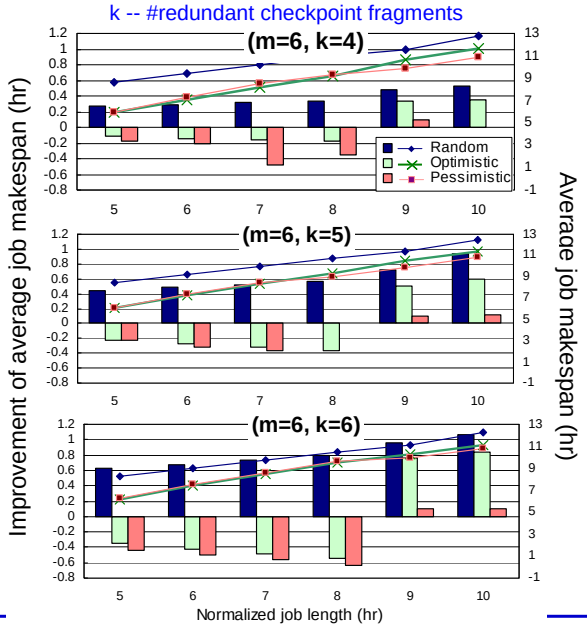
Jobs running under low resource availability (between 11 AM and 10 PM)



Average job makespans for all the 720 jobs

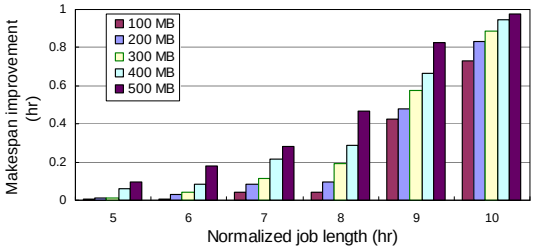
- On average, failure-aware schemes achieve an improvement of 13% and15%, respectively.
- The performance of the optimistic scheme on long jobs can be improved by using larger *k*.

m -- #basic checkpoint fragments
k -- #redundant checkpoint fragments

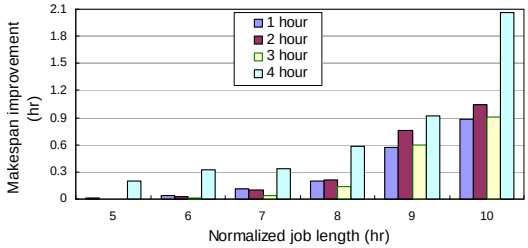


Checkpoint Intervals

Fixed set of checkpoint repositories

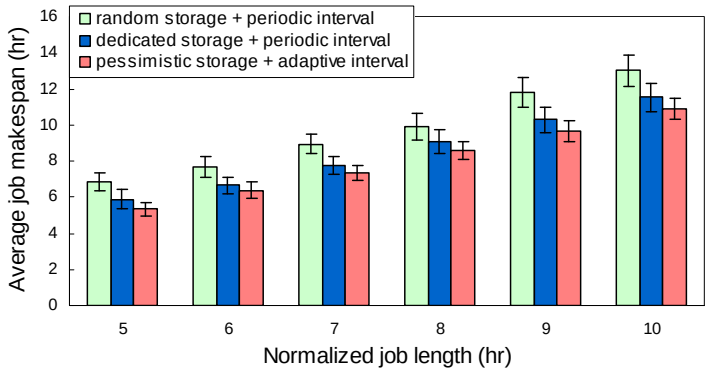


- Checkpoint periodicity = 1 hour
- Improvement increases with job length and checkpoint size



- Checkpoint size = 300 MB
- Minima for checkpointing at fixed periodicity: every 3 hours

Overall Evaluation



- Reference method
  - Dedicated storage is available 98% of the time
  - Checkpoint every 3 hours
- The 95% confidence intervals are small
- The average improvement on job makespan is 9% and 17%, respectively

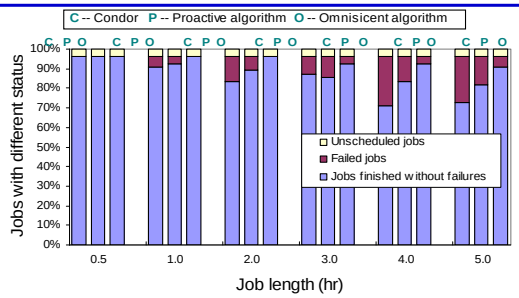
Conclusions

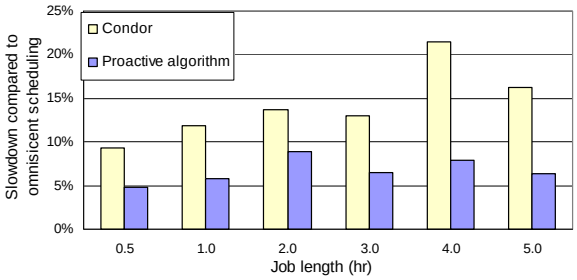
- Failure-aware checkpointing
  - Select reliable repositories from failure-prone storage
  - Adapt checkpoint intervals to the dynamism of resource availability
- Feasibility in FGCS systems
  - No dedicated hardware servers
  - Acceptable checkpoint overhead and network latency
- Improve application performance by 17% compares to methods used in production FGCS systems

Thanks!

Backup Slides

- Results of job scheduling, 27
- Ranking in the greedy algorithm, 28





Example for Ranking of Storage Hosts

Step 1. Objective function: $\min(\sum_i 4*N_i - \prod_i 50*TRS_i)$

Three storage hosts:

A > B > C

ID	N	TRS	Increment
A	4	0.6	4*4 - 50*0.6 = -14
B	4	0.3	4*4 – 50*0.3 = 1
C	3	0.2	4*3 – 50*0.2 = 11

Step 2. Objective function: $\min(16 + \sum_i 4*N_i - \prod_i 30*TRS_i)$

Two storage hosts:

C > B

ID	N	TRS	Increment
B	4	0.3	4*4 – 30*0.3 = 7
C	3	0.2	4*3 – 30*0.2 = 6