SAND2011-3733C

Fault Oblivious HPC with Dynamic Task Replication and Substitution

ISC '11, June 20, 2011

Yevgeniy Vorobeychik, Jackson Mayo, Rob Armstrong, Ron Minnich, Don Rudish
Sandia National Labs

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Outline

- Example (motivating) application domain
- The DAS Architecture
- Trading off replication and substitution
- The Grid World application
- Simulation results

Motivating Application: Distributed Value Iteration

Markov Decision Process

- A collection of states S
- Can take an action *a* in each state *s*
- Next state s_{t+1} depends on current state s_t and and current action a_t
- Receive reward (payment) $r_t = r(s_t)$
- Discount factor δ: future rewards exponentially discounted
- Goal: collect maximum expected sum of rewards

MDP + Value Iteration

- Let ***V*(s)*** be the maximum expected sum of rewards starting at state ***s***
- **Bellman optimality:**

$$V^*(s) = r(s) + \delta \max_a \sum_{s'} P^a_{ss'} V^*(s')$$

- $P^a_{ss'}$ is probability of going to state ***s'*** from ***s*** if action ***a*** is taken
- **Value iteration (starting at $V_0(s) = r(s)$):**

$$V_{n+1}(s) \leftarrow r(s) + \delta \max_a \sum_{s'} P^a_{ss'} V_n(s')$$

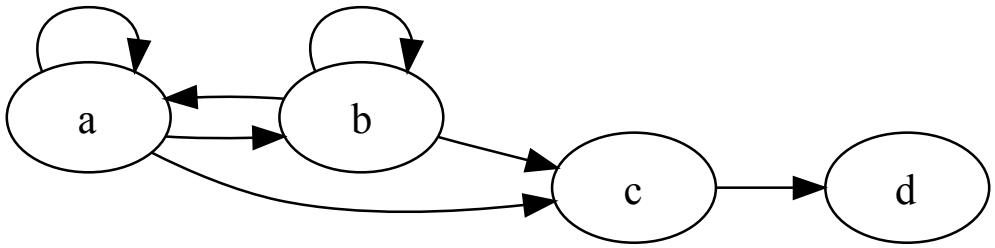
Value Iteration

- Provably converges to V^*
- Standard implementation updates values synchronously; as long as values for all states keep being updated, asynchronous variant also converges
- Requires broadcast transmission of the entire vector V_n in every step
- Requires storing the entire table of transition probabilities, etc, of total size |S||A|
- Cannot break this up into independent subproblems: no independent subsets of values

Dependency Graph

Definition

- Each node *i* represents a computational task
  - Computing $V^*(s)$ (and intermediate $V_s(s)$)
- A directed edge from *i* to *j* means that *i* depends on data generated by *j*
  - A non-zero probability of going from state *s* to *s'* means that *V(s)* and *V(s')* depend on each other
  - A task can depend on itself



Using Dependency Graph

- Can be specified/change dynamically
- Don’t need to broadcast the entire value vector: nodes can query current value from each neighbor
- Much smaller storage requirements for each node (only information about itself and neighbors)

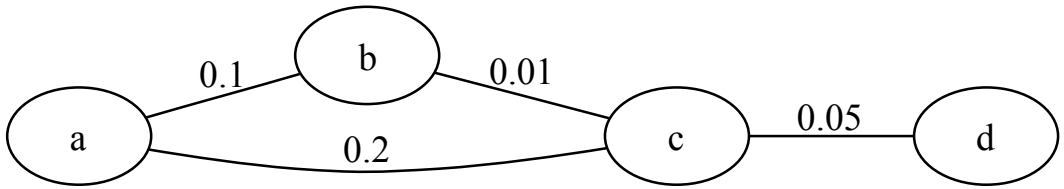
Using Dependency Graph

- Fault tolerance via replication
  - If node *i* detects that its neighbor has failed, send request to the system to restart that computation
- Checkpointing:
  - Intermediate results can be multicast to nodes that depend on task *i* (this could be the data required by these neighbors anyway)
  - If *i* fails, it can be hot-started based on intermediate results
  - Checkpointing can be (potentially) less frequent

Substitution Graph

Substitution Graph

- Edge between *i* and *j* means that computation performed at node *i* can be substituted by that at node *j* (and vice versa)
- Assume substitution relationships are reciprocal (undirected edges); not really necessary, but convenient
- Substitution can be imperfect: an edge has a weight representing an upper bound on error introduced by substitution
- Can be dynamic: graph can change "on the fly"



Transitivity of Substitution

- Note that substitution relationships need not be transitive:
  - If we only include edges with error below some threshold e, can have error(a,b) < e, error(b,c) < e, but error(a,c) > e

Fault Obliviousness via Substitution

- Suppose task *i* fails and another task *j* depends on its computation. Instead of replicating *i*, can substitute by *k*, incurring some error
- In general, substitution error depends on who uses the computation, so in general could have a different graph for each task
- Note that we can store it in a distributed way
  - each task stores only substitutions for tasks it depends on and corresponding weights

Trading Off Replication and Substitution

Notation

- Suppose errors are additive (e.g., using l_1 norm)
- w_{ijk}: error incurred by task k if computation it requires performed by i is substituted by j
- I: the set of failed tasks
- D_i: set of tasks that depend on I
- v_i = 1 iff i is substituted for (rather than replicated)
- z_{ijk} = 1 iff j substitues for i for a dependent task k (repaces k's dependence on i with j)

Mixed Integer Program Formulation

Objective: max # of substitutions (minimize # of replications)

$$\max_{v,z} \sum_{i \in I} v_i$$

Can account for the fact that subsets of tasks are allocated to different processors minimizing inter-processor communication by adding a constraint on the number of substitutions that cross processor boundaries

Only a single substitute for *i,k*

$$\sum_{j \notin I} z_{ijk} = v_i \; \forall i \in I, k \in D_i$$

Don't substitute with failed tasks

$$z_{ijk} = 0 \; \forall i, j \in I, k \in D_i$$

(unnecessary in practice)

Binary variables

$$v_i, z_{ijk} \in \{0,1\}$$

Simplified Optimization

- Previous MIP is NP-Hard to solve, probably not something you can do in real time
- Let's simplify:
  - N_i: number of tasks that depend on i
  - $w_{ij} = \max_k w_{ijk}$
  - Focus on a best substitute for a failed i, with

$$w_i = N_i \min_{i \notin I} w_{ij}$$

Simplified MIP

Objective: max # of substitutions
(minimize # of replications)

$$\max_{v} \sum_{i \in I} v_i$$

Subject to:

Error budget

$$\sum_{i \in I} w_i v_i \leq \varepsilon$$

Binary variables

$$v_i \in \{0, 1\}$$

This is just a Knapsack problem; can be approximated by a simple, fast greedy heuristic:

add tasks in increasing order of w_i until error budget is saturated

Obtaining Substitution Error Weights

- Upper bound on errors can be specified for the specific problem instance based on domain knowledge
- Empirical: observe data streams produced at different nodes and compare based on some relevant distance metric; if similar enough, add a substitution edge (and specify weight)
- Combined: specify edges a priori, but use empirical weights

Simulation Results: Value Iteration on the Grid World

Grid World

- An agent moves around the grid world
- Each cell represents a state and yields some reward *r*
- Agent can move up, down, left, right
- Resulting location is a probabilistic function of the move (agent is drunk)
  - Agent actually goes in the direction of his intended action with probability *p = 0.8*
  - Moves to any other adjacent location with equal probability
  - Grid boundaries are “walls” (no wrap around)

Example grid world

Value Iteration, Substitution Errors

- **Set discount factor to 0.95**
- **Stopping criterion at 0.001 (stop once values are not changing by more than this)**
- **Values of neighboring states *s* and *s'* are close substitutes for one another; can bound error of such substitution (w_{ij}):**

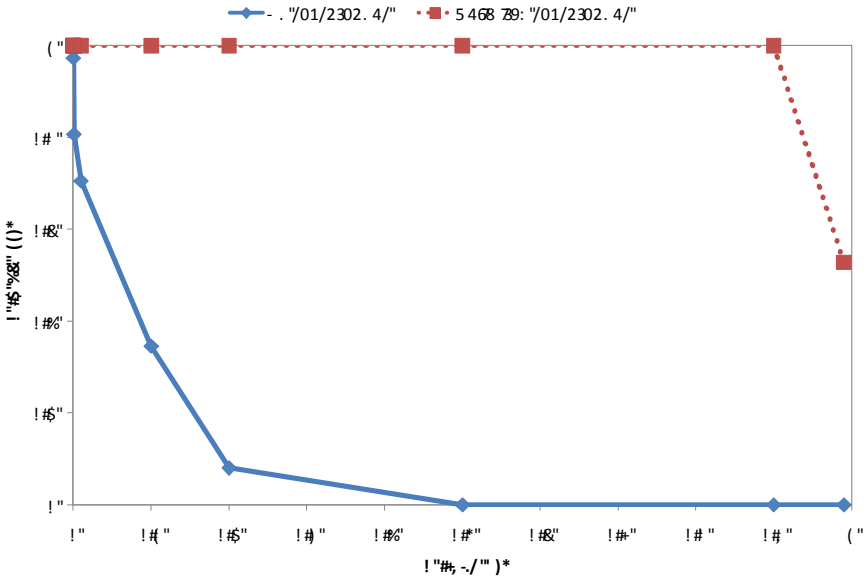
$$|V(s) - V(s')| \leq \frac{1 - \delta p}{1 - \delta}$$

Simulation Setup

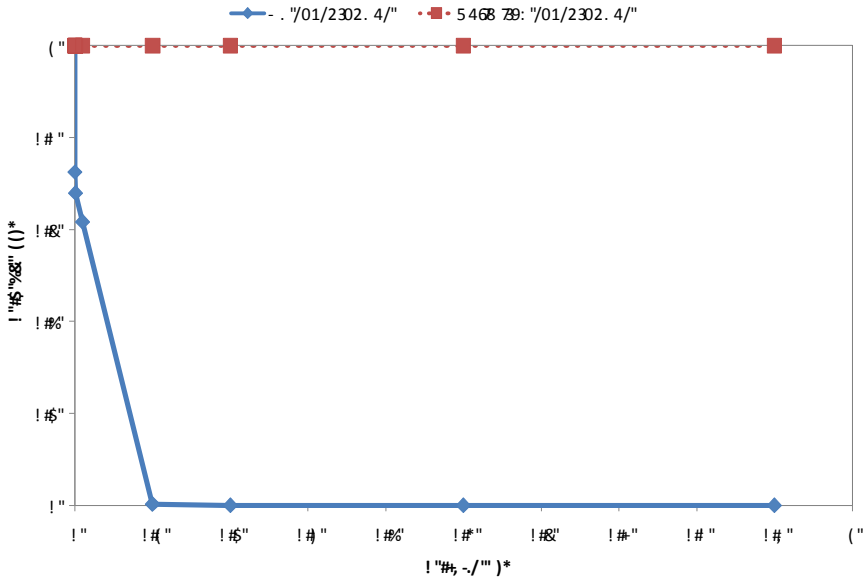
- Simulate a cluster; allocate a task per cluster node; run 100 iterations
- Nodes can fail and are fixed probabilistically (independently); ½ of all processors are functional on average; initialized to steady state
- Number of tasks is ½ of total number of nodes (saturate expected number of functioning processors)
- Compare pure replication (always replicate failed tasks) to pure substitution (always substitute)

Resilience to Failures

10x10 grid



50x50 grid

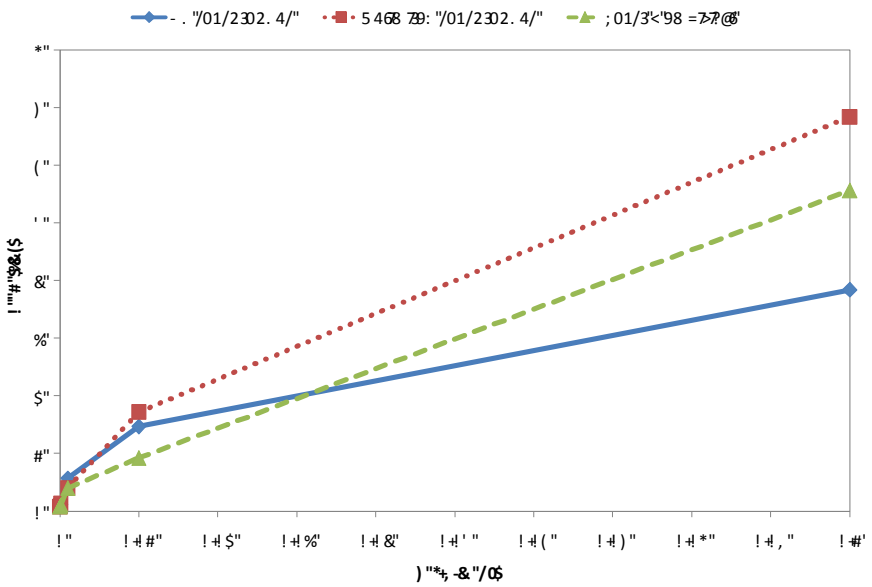


Advantage of substitution greater on a larger problem

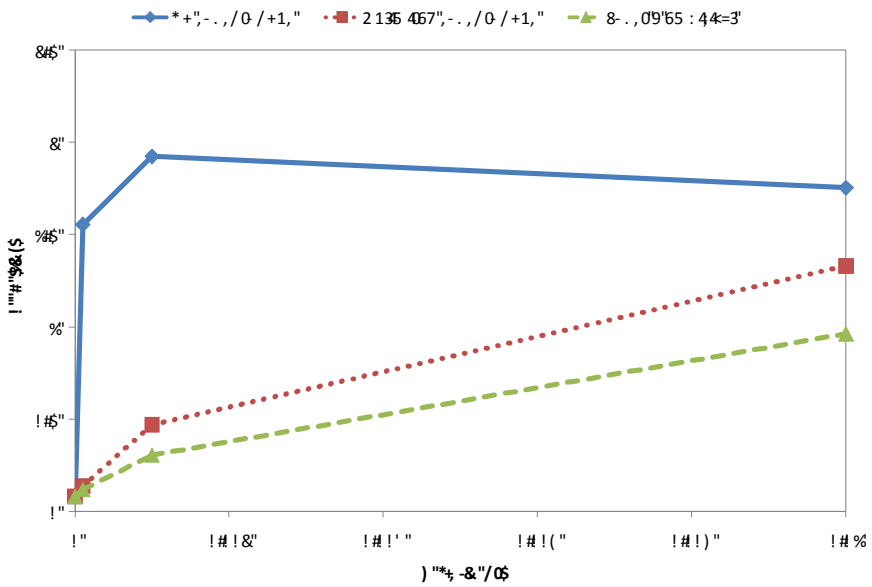
Sandia National Laboratories

Computation Error

10x10 grid



50x50 grid



Empirical substitution is best when individual node failure probability is low; advantage greater on the larger problem

Conclusions

- Novel architecture for fault tolerance/obliviousness using a combination of task dependency and substitution graphs
- Optimization problem formulates tradeoffs between replication and substitution; can approximately solve a simplified problem in real time
- Simulation results suggest that ability to substitute tasks is advantageous for resilience and keeps errors down