Infrastructure Reliability & Security Management using Partially Observable Markov Decision Processes

William H. Sanders and Kaustubh R. Joshi

Information Trust Institute and
Coordinated Science Laboratory,
University of Illinois at Urbana-Champaign
Urbana, IL

Matti A. Hiltunen and Richard D. Schlichting

AT&T Labs - Research
Florham Park, NJ

IFIP Working Group 10.4
June 29-30, Annapolis, Maryland

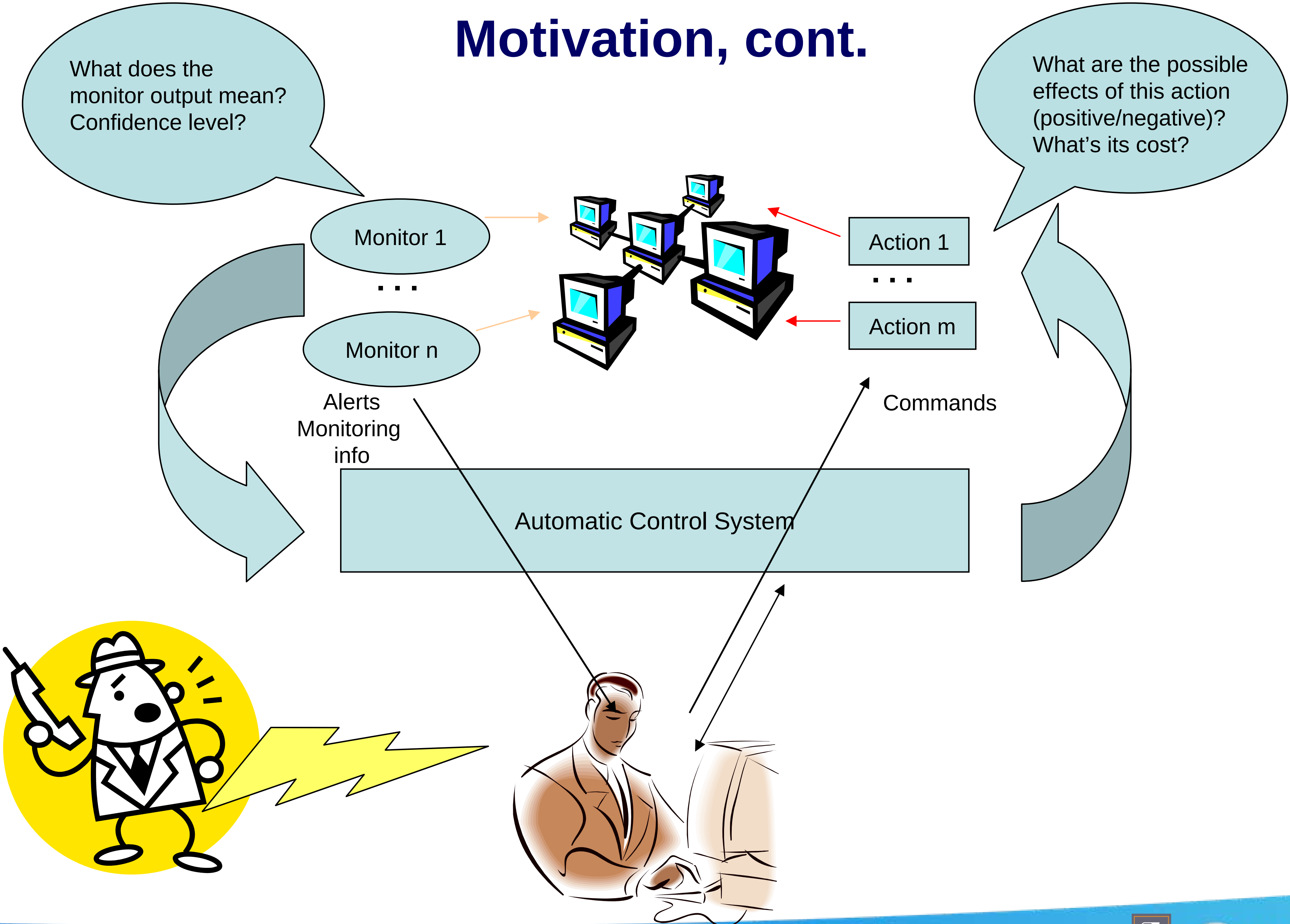
Outline

- Motivation
- Driving Application
  - A problem looking for a (better) solution
  - Other infrastructures with similar problems
- Model-based solution
  - Probabilistic diagnosis
  - POMDP-based recovery
  - Stability and performance properties
  - Results
- Future work
- Conclusions

Motivation

- The need for automated system management
  - Driving factors speed of response, cost, and amount of data
- System management an example of adaptation.
- Drivers of change
  - Failures and attacks
  - Changing workloads and requirements
  - Changing resources
- Dealing with change
  - Recovery – rapid response crucial
  - Rejuvenation - preventive maintenance and reconfiguration
  - Only different in the types of indicators used

Motivation, cont.



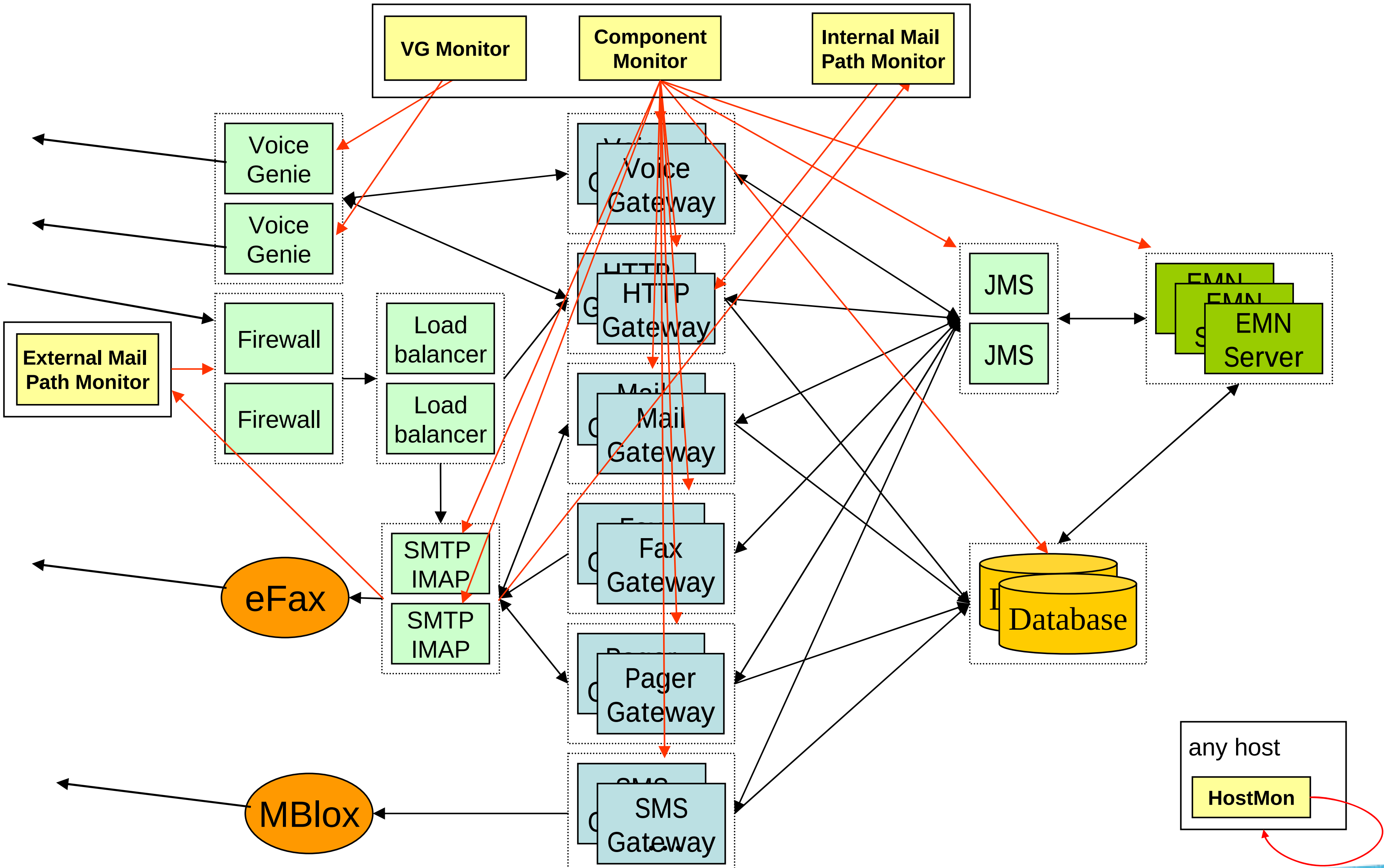
Automatic Systems Management

- Triggers, actions, and metrics
- Fundamental cost benefit tradeoff
  - When is change needed, what benefits does it bring?
  - Simplest example – does adaptation take system to a “good” state?
  - Need a way to encode some operator knowledge (e.g., which actions may correct a problem)
  - Need metrics (cost/rewards) to perform this automatically

Driving Application

- *Problem*: Monitoring and operator alerting for a complex internet-based system
- Home grown + COTS components:
  - Firewalls, load balancers, web servers, JMS servers, databases, Voice Genie, SMTP/IMAP servers,..
  - Network elements: routers, switches, links
  - External services
- Different independent monitors for some individual components and for end to end service functionality.
- Problems:
  - Lots of operator alarms (one problem, multiple alarms)
  - False positives
  - Poor localization (i.e., what is the real problem)
  - Not great fault coverage
- Goals: Make things better

Example Application: AT&T's EMN

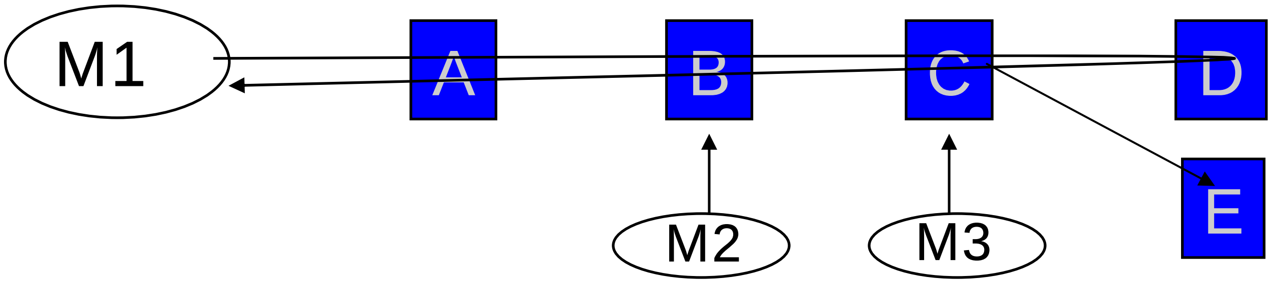


Previous Solution

- Collect the outputs of all the monitors into a centralized syslog.
  - Disable direct operator alerting from the individual monitors.
- A *MasterMonitor* program continuously reads the log, forms an estimate of the system state, and alerts operators when necessary.
  - Various heuristics used to combine information.
  - Use passage of time to deal with false positives.
  - Combine outputs from multiple monitors to eliminate possibilities (i.e., narrow down the faulty component)

Lessons Learned

- Diagnosis can be difficult: Which component is faulty?



Maybe: A or D

However, could be any, because M1 and M2/M3 may not detect the same fault types

Complexity of coding such rules was getting out of hand

Key Observations

- Different system monitors detect different types of problems => fault hypotheses
  - Monitor outputs and recovery actions can be characterized in terms of these fault hypothesis
- Monitors do not always detect the problem => fault coverage (probability)
- “Path monitor” concept – a monitor tests a path through the system
- Need for a general methodology for monitor output fusion
- The presence of failures in the system can be only deduced based on the monitor outputs (= observations)
- Typically no absolute knowledge of faulty component => recovery actions must be used to improve diagnosis
- Performing more monitoring is often a good action to take
- Automated system must know when to give up

Similar Problem Areas

Network monitoring and automatic recovery:

- Similar problems: lots of different types of components (routers, links,..), faults on different levels (optical, IP, VPN, ..)
- Some automatable recovery actions: routing changes, restarts, …
- Some work on fault diagnosis – shared link risk group (SLRG – NSDI 05)
- Note that automatic recovery action may be simply to monitor more, or run more detailed monitoring specific to the anticipated problem

Similar Problem Areas, cont.

Security: conceptually a good match:

- IDSs = monitors, system attack status often unknown,
- A range of actions (e.g., port blocking, routing to a scrubber, routing to a black hole)
- Extra challenges: attacker may figure out a way to bypass monitor/IDS

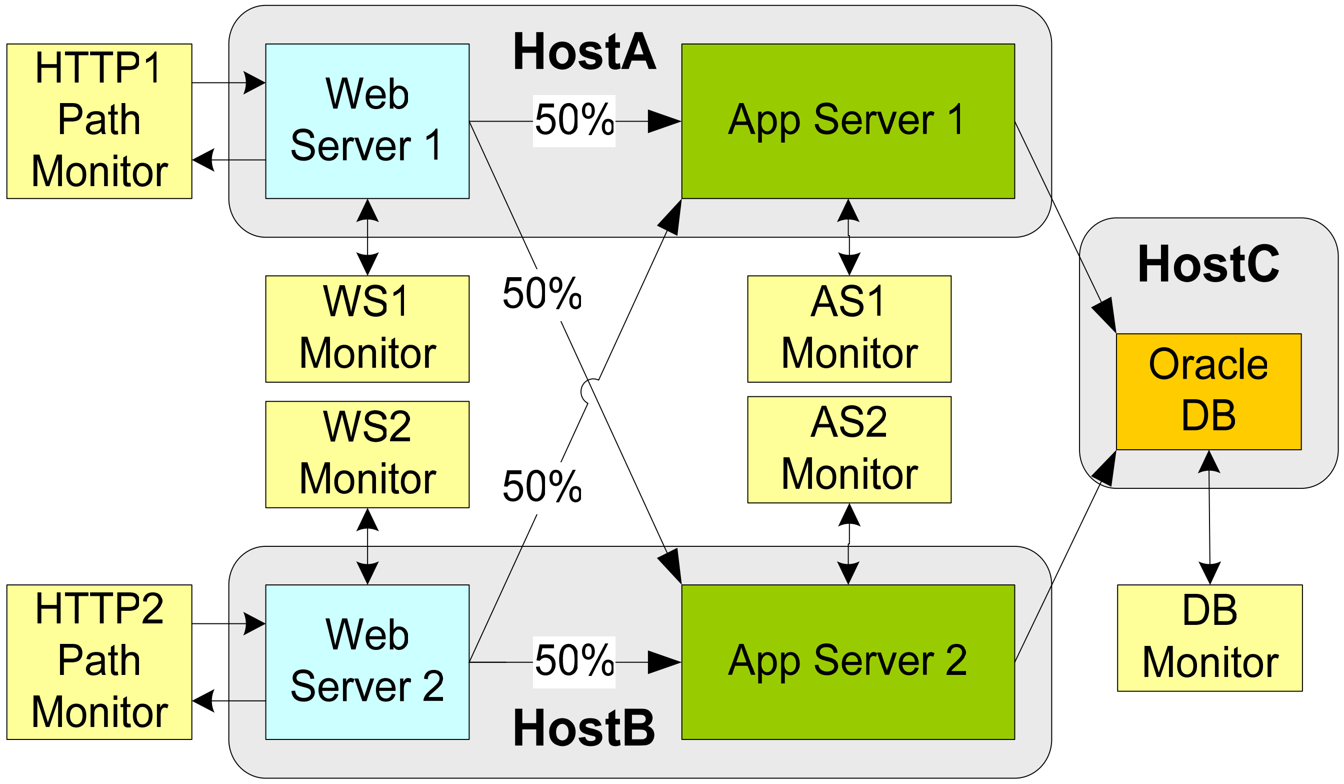
Challenges in Recovery

- Opaqueness makes diagnosis difficult
  - Multiple tiers span administrative domains and technology layers
  - Poor localization, false positives and negatives, imperfect coverage
  - Each monitoring technique has different strengths/limitations
  - Result: uncertainty about true system state
- Multiple choices of recovery actions
  - Varying cost
    - Restart component vs. reboot host
    - Act now or wait until later?
    - Ordering constraints between component restarts
  - Varying benefit - not all failures are equal
    - Different components are valued differently depending on their customer impact.
- What if the automated system becomes unstable?
  - Ad-hoc vs. theoretically founded approaches

Abstracted Problem

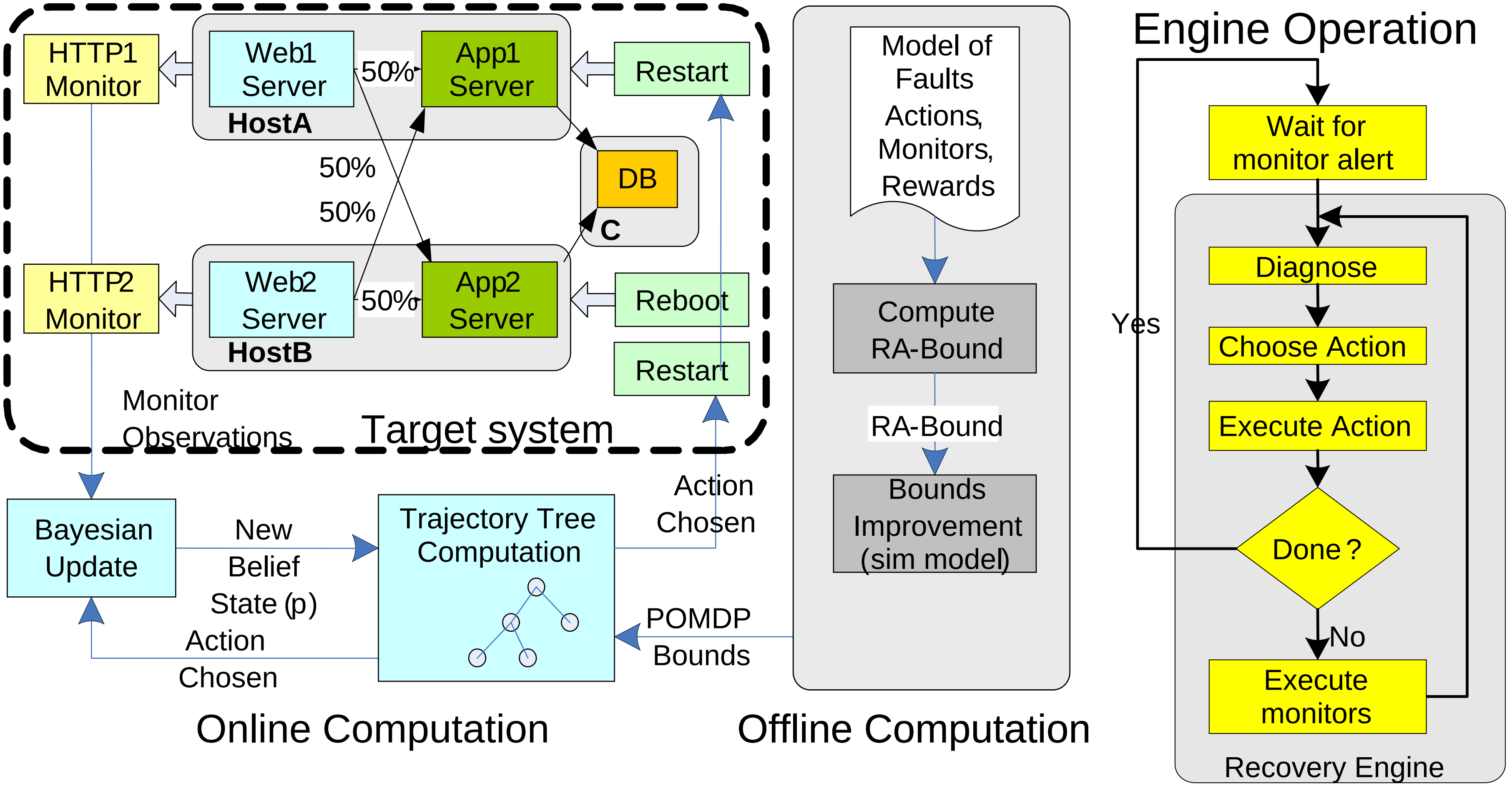
- Some simplifying assumptions:
  - Monitors can be invoked at will
  - Monitor output = {true, false}
  - Only one fault hypothesis is true at a time
  - Constant fault coverage for each monitor (i.e., no change over time)
  - No transient failures
- Simplified example system: 3-tier e-commerce system

Example: An E-commerce System



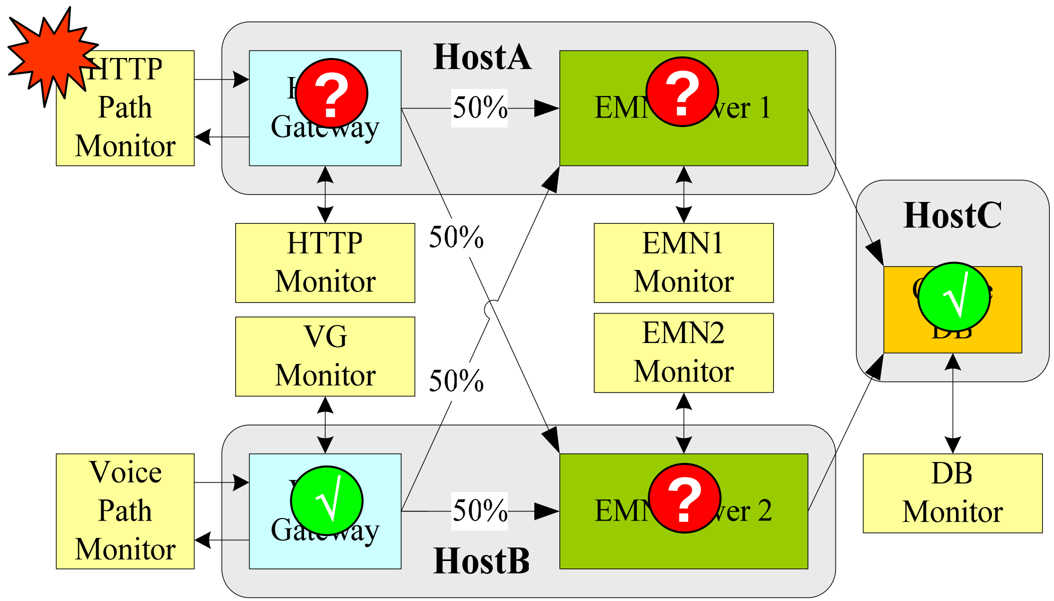
- Fault models: fail-silent (crash), non fail-silent (zombie) faults
- Recovery Actions: restart component, reboot host.
- Individual component monitors: only detect crashes
- End-to-end path monitors: detect crashes and zombies but poor localization
- Recovery Cost: fraction of “lost” requests (i.e. user-perceived availability)

Recovery Engine Architecture



- Action that maximizes value function tree is chosen at each step
- What to use for remaining cost at the leaves of the tree?
  - Zero cost, heuristic cost, bound?

Probabilistic Bayesian Diagnosis



- Precise diagnosis often impossible due to monitor limitations
- Use Bayes rule to compute "diagnosis vector" $\{P[fh_1], \cdots, P[fh_n]\}$
  - Each entry: probability of *fh* given current monitor outputs
  - Using monitor coverage models $P[m|fh]$ and prior diagnosis
  - If no prior knowledge of which fault, use $P[fh]=1/|FH|$
  - Keep track of commonly occurring faults to choose better priors

Monitor Models

- Need to know coverage: P[*m*|*fh*]
- Dependency graph based
  - Probability of touching failed node in a request graph
- Queuing network based
  - Probability of observed response time, load
  - Statistical test
- Statistically learned models in general

POMDP Formulation for Recovery

- A POMDP is a tuple $(S,A,O,p(s'|s,a),q(o|s,a),c(s,a))$
  - States (S): which fault (or null fault) has occurred
  - Observations (O): monitor outputs $\{o_m\}$
  - Transition function $p(s'|s,a)$: effect of recovery action on system and fault state
  - Observation probabilities $q(o|s,a)$: probability that o is generated (monitor models)
  - Cost Function $c(s,a)$: recovery cost, e.g., availability, requests lost/denied etc

- System evolution
  - $(s_0,a_0,o_0,\ldots s_n,a_n,o_n)$
  - But controller can't see s – it tracks "belief state"
  - Belief state $\pi = [\pi(s_0),\ldots,\pi(s_n)]$: state occupancy probability vector (i.e., diagnosis vector)

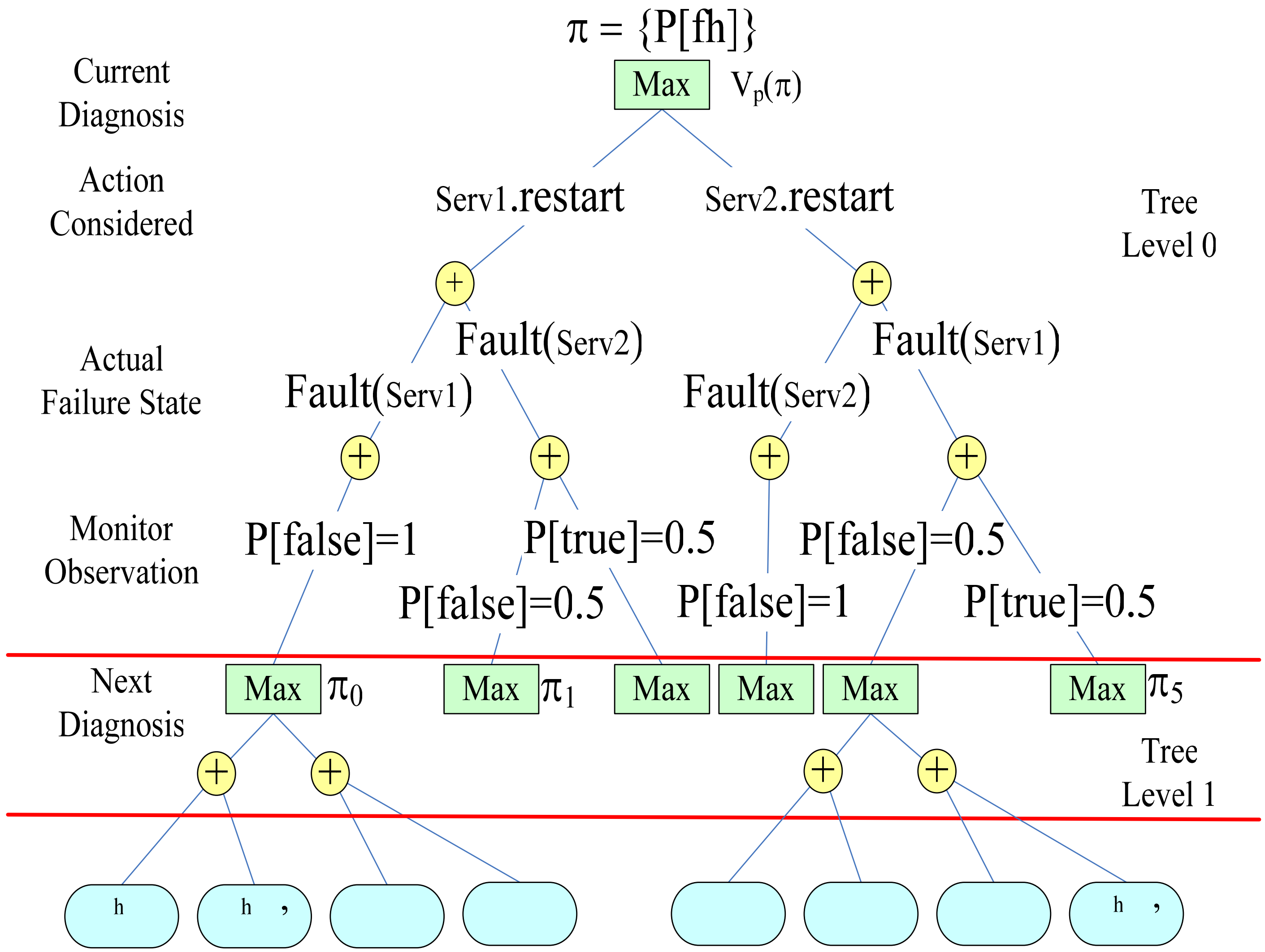
Optimal Value Functions

- Policy ρ specifies what action to take in each belief state
  - Optimal policy ρ^* minimizes mean accumulated cost starting from all belief states
  - ρ^* is Markovian in belief state (i.e. current diagnosis vector)

- Optimal ρ^* computation
  - Bellman dynamic programming recursion
  - $C_m(\pi) = min_a\{c(s,a) + \mathcal{H}_{s'}[C_m(\pi')]\}$
  - $p'(\pi'|\pi,a) = \sum_s q(o|s,a) \sum_{s'} p(s|s',a)\pi(s')$ if π'=BayesNextBelief(π,a,o)
    = 0 otherwise
  - $c'(\pi,a) = \sum_s c(s,a)\pi(s)$

- Tractability is a problem.
  - Dynamic programming defined over all π
  - There could be infinite π even for trivial S!
  - Exact techniques scale only up to few thousand states

Finite Depth Online POMDP Solution



Leaves are assigned heuristically chosen or bounded cost

Recovery Engine Guarantees

- Desired Guarantees:
  - Safety: recovery engine does not execute unsafe actions
  - Guaranteed recovery: engine does not terminate before recovery is successful (can only be guaranteed w.r.t. model)
  - Finite termination: recovery terminates in a finite amount of time
  - Optimal performance (ideal): recovery cost is minimized
  - Performance guarantee (practical): recovery cost may not be optimal, but is lower than a promised value
- POMDP based recovery engine using finite depth solution
  - Safety can be ensured at model level by disabling dangerous actions
  - Heuristic value at leaves: we can make no guarantees
  - Lower bounds of true value: probabilistically guaranteed recovery, finite termination, average performance guarantee

Value Function Lower Bounds: RA-Bound

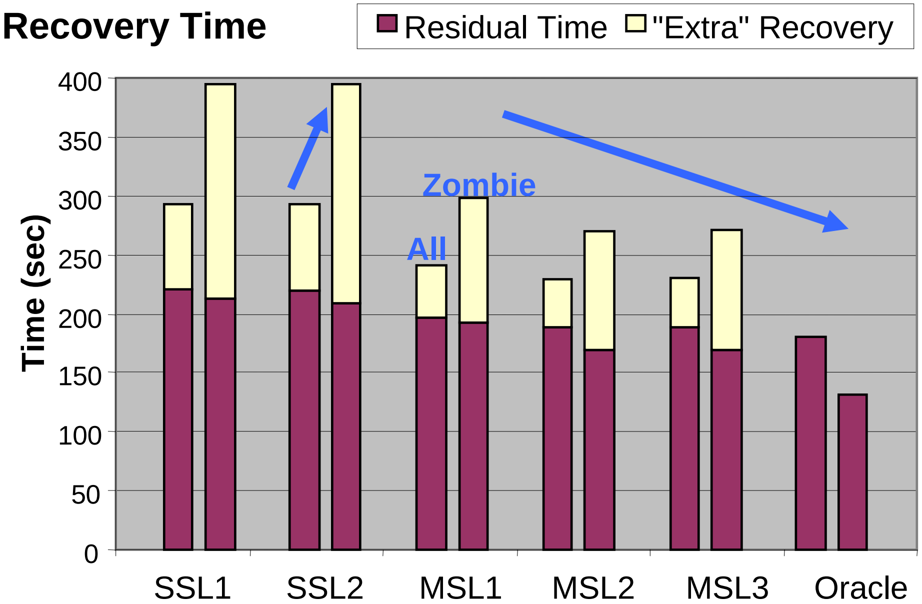
- Previously: Bounds on discounted rewards
  - Discounted reward: $V(\pi) = \max_a\{r(s,a) + \beta\, H_{s'}[V(\pi')]\}$
  - Previous techniques: BI-POMDP, blind action
  - Always finite – even when controller never terminates!
  - Difficult to determine "good" β – weak relation to reality
- New (DSN'06): Bounds on undiscounted accumulated reward
  - Value function may be infinite
  - BI-POMDP, blind action not always finite even for finite valued recovery models
  - We develop a new bound (RA-bound) and conditions under which it works for recovery models
  - Can evaluate risk of terminating recovery too early

Key Practical Benefits

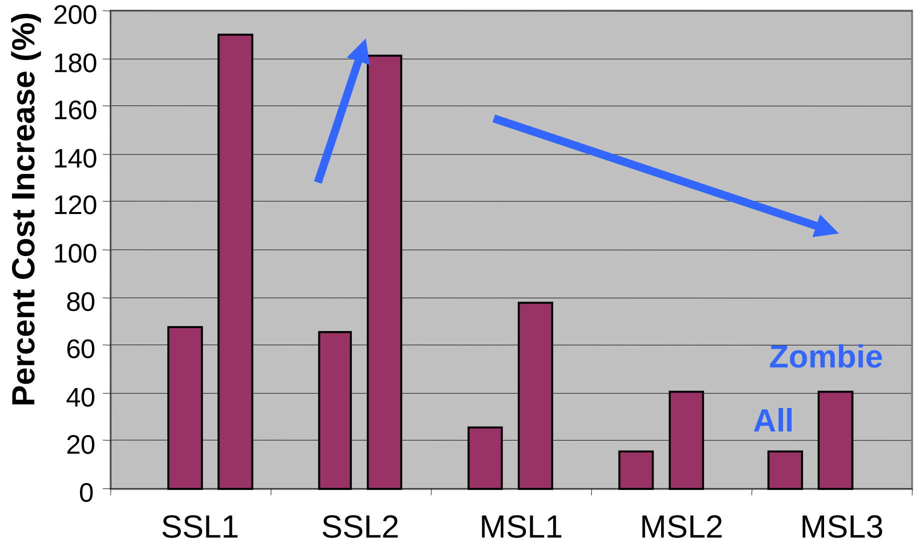
- Model based – allows separation of concerns monitoring and recovery during specification
- Reward based recovery considers both cause and impact – precise root cause identification may not be critical
- Sequential recovery – natural way to deal with mistakes
- Ability to look multiple time-steps ahead – knows when to wait for additional information
- Formal framework – provides strong guarantees about stability and goodness of adaptation

Greedy vs. POMDP (heuristic): Per-Fault Metrics

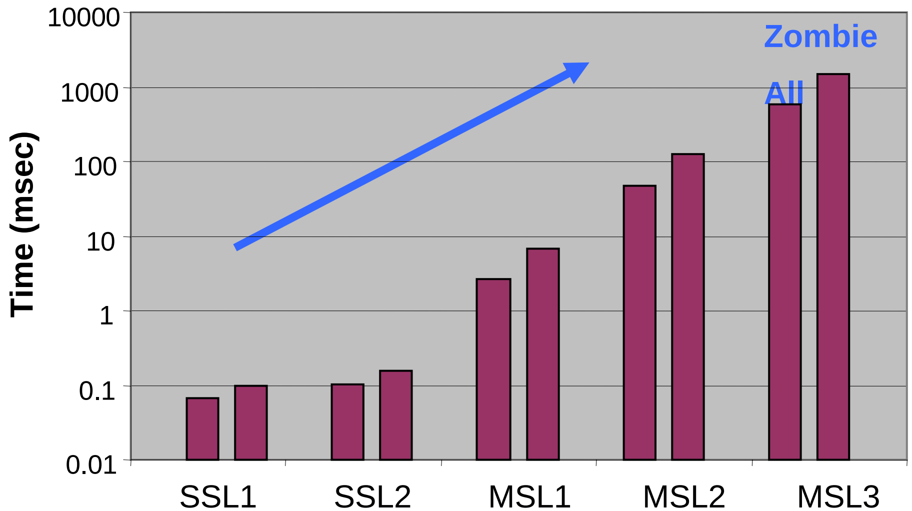
Recovery Time



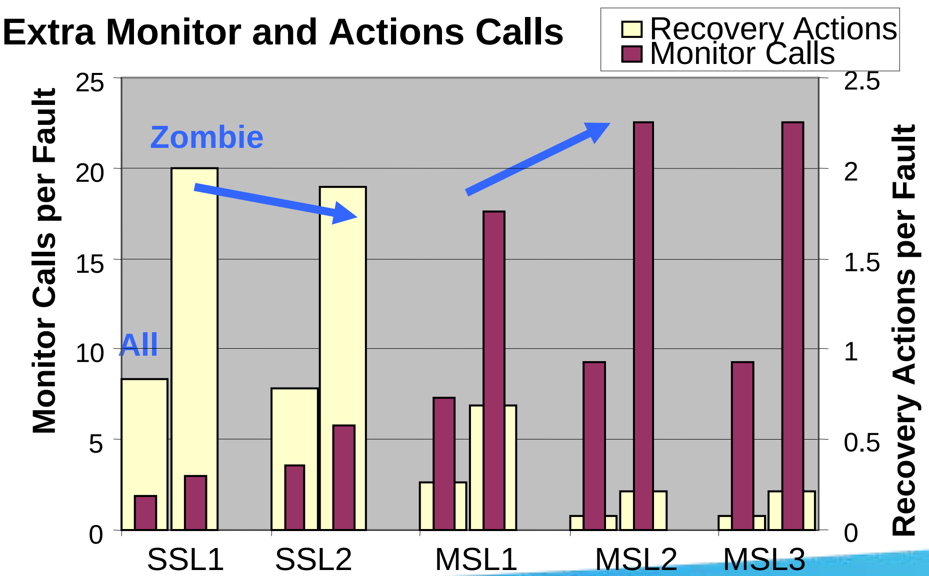
Cost: % Increase of lost requests over oracle



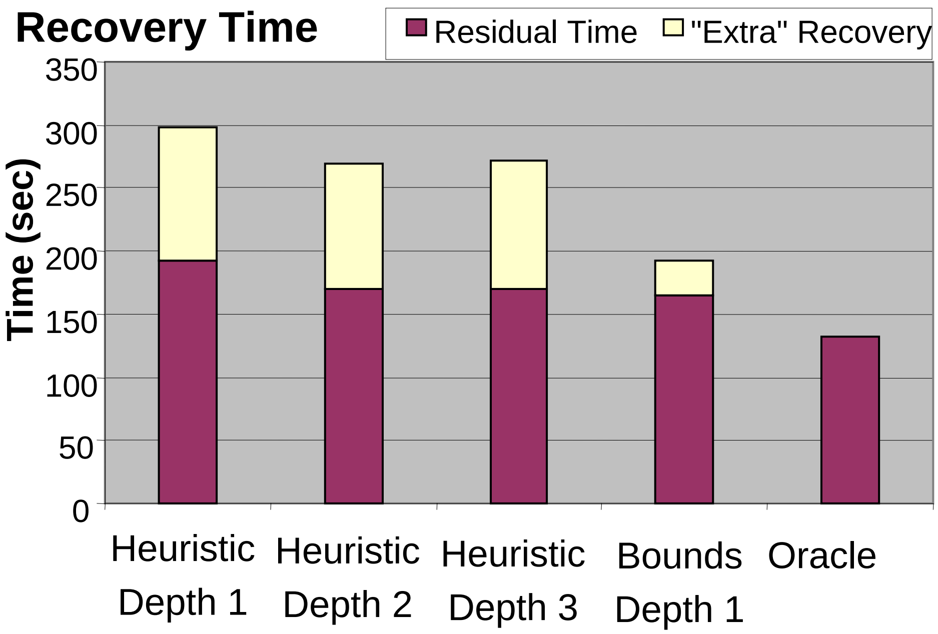
Algorithm Running Time (msec)

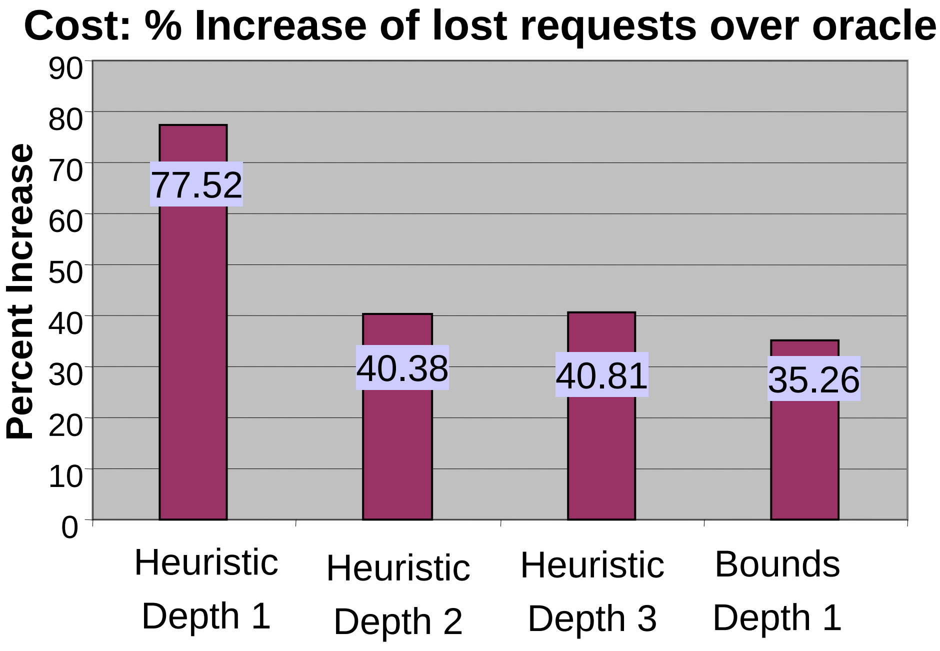


Extra Monitor and Actions Calls

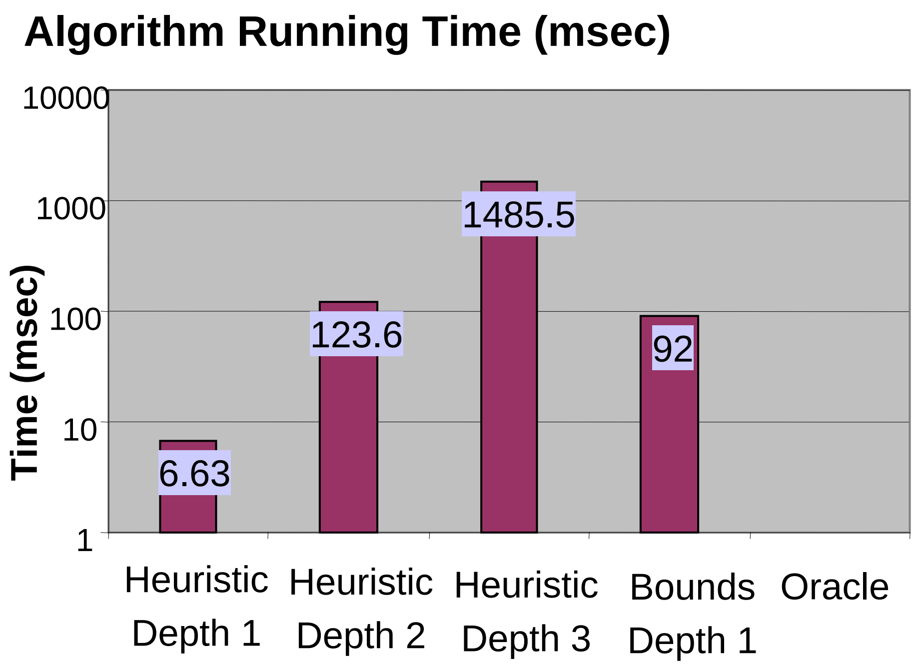


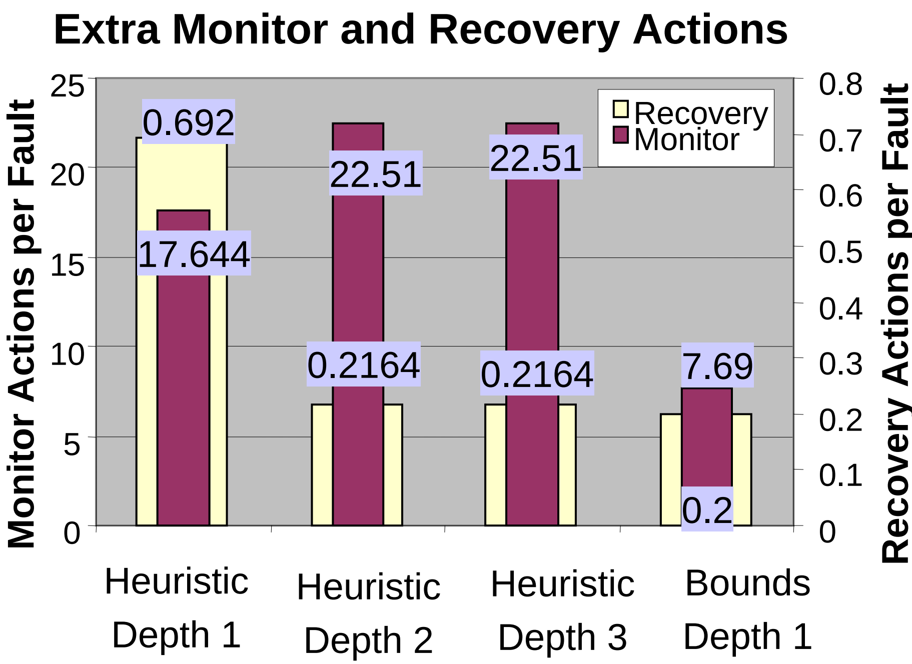
POMDP (heuristic vs. bounds, zombie only)





POMDP (heuristic vs. bounds, zombie), cont.





Future work

Model extensions:

- Continuous time: allowing independent system evolution

Engine extensions:

- Dealing with real monitors' outputs: textual, non-standard
  - Combine rule-based and probabilistic reasoning
  - Rules good when no uncertainty of the problem
- Some monitors cannot be invoked at will
  - Must wait for the "next scheduled" output
  - Sometimes monitors only give failure alarms but do not report recovery - Absence of alarm for a period of time = all OK
- System specification in general format (XML)
  - Components, their relationships, monitors, fault hypotheses, coverage, allowed actions, ..
  - Different system configurations
- Load-aware monitors for performance failures (queuing model based)

Conclusions

- A model-based solution for system diagnosis and automatic recovery develop based on needs identified in a real system (SRDS 05)
- New technique developed for solving models efficiently and accurately (DSN 06)
- Extensions underway to address issues in realistic systems
- Other application areas possible; evaluation part of future work (I could tell you, but Matti would kill me)

Questions?

Automatic Recovery using Bounded Partially Observable Markov Decision Processes

William H. Sanders and Kaustubh R. Joshi
Information Trust Institute and
Coordinated Science Laboratory,
University of Illinois at Urbana-Champaign
Urbana, IL

Richard D. Schlichting and Matti A. Hiltunen
AT&T Labs - Research
Florham Park, NJ

IFIP 10.4 Infrastructure Security and Reliability Management Workshop
June 29-30, Annapolis, Maryland