Distributed Diagnosis of Failures in a Three Tier E-Commerce System

Gunjan Khanna, Ignacio Laguna, Fahad A. Arshad, and Saurabh Bagchi
Dependable Computing Systems Lab (DCSL)
School of Electrical and Computer Engineering
Purdue University

Motivation

- Connected society of today has come to rely heavily on distributed computer infrastructure
  - ATM networks, Airline reservation systems, distributed internet services
- Increased reliance on Internet services supported by multi-tier applications
  - Cost of downtimes of these systems can run into several millions of dollars
    - Financial brokerage firms have a downtime cost $6.2M/hr (Source: IDC, 2005)
- Need a scalable real-time diagnosis system for determining root cause of a failure
- Identify the faulty elements so that fast recovery is possible (low MTTR), giving high availability
Challenges in Diagnosis for E-Commerce Systems

- **Diagnosis**: Protocol to determine the component that originated the failure
- There is complex interaction among components in three tier applications
  - Components belong to one of three layers - web, application tier and database
  - Not all interactions can be enumerated *a priori*
  - Large number of concurrent users
  - Rate of interactions is high
- Errors can remain undetected for an arbitrary length of time
  - An error can propagate from one component to another and finally manifest itself as a failure
  - High error propagation rate due to high rate of interactions

Solution Approach: Monitor

- **Non intrusive**: observe messages exchanged and estimate application state transitions
- **Anomaly based**: match against a rule base
- **Online**: fast detection and diagnosis
- Treat application entities (or components) as Black-box
Tracking Causal Dependencies

- Messages between the components are used to build a Casual Graph (CG)
- Vertices are components, and edges are interactions between components
- Intuition: A message interaction can be used to propagate an error

Components interacting by message passing

![Diagram showing components A, B, C, and D with message IDs and logical times.]

<table>
<thead>
<tr>
<th>Message ID</th>
<th>Logical Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>L1</td>
</tr>
<tr>
<td>2</td>
<td>L2</td>
</tr>
<tr>
<td>3</td>
<td>L3</td>
</tr>
</tbody>
</table>

Aggregate Graph

- We bound the size of the Casual Graph
  - Unbounded CG would lead to long delays in traversing the CG during diagnosis leading to high latency
  - However, complete purging of the information in the CG can cause inaccuracies during the diagnosis process
- We aggregate the state information in the Casual Graph at specified time points and store it in an Aggregate Graph (AG)
- The AG contains aggregate information about the protocol behavior averaged over the past
  - AG is similar to CG in the structure (a node represents a component and a link represents a communication channel)
  - AG maintains historical information about failure behavior of components and links
Diagnosis in the Presence of Failures

- When a failure is detected in a node $n_f$, Monitor checks which other components have interacted with $n_f$ in the CG
  - Bounded search space due to the bounded nature of CG
- The diagnosis algorithm is then triggered

Diagnosis Tree

- A tree rooted at the $n_f$ (where the failure is detected) is built from CG: Diagnosis Tree
  - Nodes which have directly sent a message to $n_f$ are at depth 1
  - Nodes which have directly sent a message to a node at depth $h$ are at depth $h+1$
Diagnostic Tests

- Diagnostic tests are specific to the component and its state
  - The state is as deduced by the Monitor
  - Example: After receiving 10 KB of data, did the component send an acknowledgement?
  - Example: After receiving a database update due to a transaction, did the update complete within 5 seconds?

- No active probing of the components with tests
  - State of the component may have changed during the error propagation
  - Probing introduces additional stress on the components when failures are already occurring

- Tests may not be perfect
  - They are probabilistic in nature

PPEP: Path Probability of Error Propagation

- Definition: $PPEP(n_i, n_j)$, the probability of node $n_i$ being faulty and causing this error to propagate on the path from $n_i$ to $n_j$, leading to a failure at $n_j$

- $PPEP$ depends on
  - Node Reliability of the node which is possibly faulty
  - Link Reliability of links on path
  - Error Masking capability of intermediate nodes on path

Example: $PPEP(B, D)$
Node Reliability

- **Node Reliability** \((n_r)\): quantitative measure of the reliability of the component
- It is obtained by running the diagnostic tests on the states of the components
  - Coverage of the component \(c(n) = \frac{\text{#tests that succeeded}}{\text{#tests}}\)
- PPEP is inversely proportional to node reliability
  - More reliable node, therefore less likelihood of originating the chain of errors

Link Reliability

- **Link Reliability** \((l_r(i,j))\): quantitative measure of the reliability of the network link between two components
- \(l_r(i,j) = \frac{\text{number of received messages in receiver } n_j}{\text{number of sent messages in sender } n_i}\)
- PPEP is proportional to link reliability
  - Reliable links increase probability of the path being used for propagating errors
Error Masking Capability (EMC)

- **Error Masking Capability** ($e_m$): quantitative measure of the ability of a node to mask an error and not propagate it to the subsequent node on the path.
- EMC of a component depends on the type of error, e.g., syntactical or semantic errors:
  - A node may be able to mask all “off by one” errors but not an error which leads to a higher rate stream.
- PPEP is inversely proportional to the EMC of node in the path:
  - The intermediate nodes are less likely to have propagated the error to the root node.

Calculating PPEP for a Path

- Example: Calculate PPEP from B to D

\[
PPEP(B, D) = (1 - n_r(B)) \cdot l_r(B, C) \cdot (1 - e_m(C)) \cdot l_r(C, D)
\]

- For a general path,

\[
PPEP(n_1, n_k) = (1 - n_r(n_1)) \cdot l_r(1, 2) \cdot (1 - e_m(n_2)) \cdot l_r(2, 3) \cdot \ldots \cdot l_r(i, i+1) \cdot (1 - e_m(n_{i+1})) \cdot l_r(i+1, i+2) \cdot \ldots \cdot l_r(k-1, k)
\]
Overall Flow of the Diagnosis Process

Components interacting

Monitor detects failure

Start Diagnosis Process

Create Diagnosis Tree (DT)

Traverse DT down from the root

Apply Diagnostic Tests (compute node reliabilities)

Calculate PPEP for each node

Top-k PPEP nodes are diagnosed

Node reliab.

Link reliab.

EMC

Experimental Testbed

- Application: Pet Store (version 1.4) from Sun Microsystems
- Pet Store runs on top of the JBoss application server with MySQL database as the back-end providing an example of a 3-tier environment
- A web client emulator which generates client transactions based on sample traces (written in Perl)
- For the mix of client transactions, we mimic the TPC-WIPS0 distribution with equal percentage of browse and buy interactions
  - A web interaction is a complete cycle of communication between the client emulator and the application
  - A transaction is a sequence of web interactions. Example: Welcome page → Search → View Item details
Logical Topology

- Web client emulator
- External Failure Detector
- J2EE Application Server
  - Web Tier
    - servlet
    - JSP
  - EJB Tier
    - EJB
- Internal Failure Detector
- Database

Monitor Configuration
- The Monitor is provided an input of state transition diagrams for the verified components and causal tests

Example Rules
- S0 getData 1 S2 return getData 1
- S0 getExpiryMonth 1 S1 return getExpiryMonth 1
- S0 getExpiryData 1 S3 return getExpiryData 1
Comparison with Pinpoint

- Pinpoint (Chen et al., DSN ’02, NSDI ’04) represents a recent state-of-the-art black-box diagnosis system
  - Well explained and demonstrated on an open source application
  - We implement the Pinpoint algorithm (online) for comparison with the Monitor’s diagnosis approach

- Summary of Pinpoint algorithm:
  - Pinpoint tracks which components have been touched by a particular transaction
  - Determine which transaction has failed
  - By a clustering algorithm (UPGMA), Pinpoint correlates the failures of transactions to the components that are most likely to be the cause of the failure

Performance Metrics

- **Accuracy**: a result is accurate when all components causing a fault are correctly identified
  - Example, if two components, $A$ and $B$, are interacting to cause a failure, identifying both would be accurate
  - Identifying only one or neither would not be accurate
  - Example: if predicted fault set is \{A, B, C, D, E\}, but faults were in components \{A, B\} $\rightarrow$ accuracy still 100%

- **Precision**: penalizes a system for diagnosing a superset
  - Precision is the ratio of the number of faulty components to the total number of entities in the predicted fault set
  - Above example: precision = 40%
  - Components \{C, D, E\} are false positives
Fault Injection on Different Components

- We use 1-component, 2-component and 3-component triggers

- **1-component trigger**: every time the target component is touched by a transaction, the fault is injected in that component

- **2-component trigger**: a sequence of 2-components is determined and during a transaction, the last component in the transaction is injected
  - This mimics an interaction fault between two components
  - Both components should be flagged as faulty

- **3-component fault** is defined similarly as in 2-component

Results for a 1-Component Fault Injection

- Both can achieve high accuracy but Pinpoint suffers from high false positive rates

- In Pinpoint, two different accuracy values can be achieved since a given precision value is achieved for two different cluster sizes

- The latency of detection in our system is very low
  - Thus, the faulty component is often at the root of the DT in the Monitor.
Results for a 2-Component Fault Injection

- Performance of the Monitor declines and Pinpoint improves from the single component fault
  - Monitor still outperforms Pinpoint
- Monitor gets less opportunity for refining the parameter values ⇒ the PPEP calculation is not as accurate as for the single component faults

Results for a 3-Component Fault Injection

- Performance of the Monitor declines and Pinpoint improves from the single and two component faults
- Monitor still outperforms Pinpoint
- Pinpoint maximum average precision value ~ 27%
  - Attributed to the fact that more number of components causes selected transactions to fail ⇒ better performance by the clustering algorithm
- Monitor declines because of the same reason as in 2-component faults
Latency Comparison

- The Monitor has an average latency of 58.32 ms with a variance of 14.35 ms.
- After 3.5 minutes the accuracy and precision of Pinpoint increase with latency.
- Pinpoint takes as input the transactions and corresponding failure status every 30 seconds during a round.
  - Pinpoint runs the diagnosis for each of these snapshots taken at 30 second intervals.

Related Work

- **White and Black box systems**
  - White box where the system is observable and, optionally, controllable.
  - Black box where the system is neither.

- **Debugging in distributed applications**
  - Work in providing tools for debugging problems in distributed applications.
  - Performance problems: "Performance debugging for distributed systems of black boxes,” SOSP ‘03.
  - Misconfigurations: "PeerPressure” ACM SIGMETRICS ‘04.

- **Network diagnosis**
  - Root cause analysis of network faults models.
  - “Shrink,” ACM SIGCOMM ‘05.
  - "Correlating instrumentation data to system states," OSDI ‘04.

- **Automated diagnosis in COTS systems**
  - Automated diagnosis for black-box distributed COTS components.
  - "Automatic Model-Driven Recovery in Distributed Systems," SRDS ‘05.
Conclusions

• We presented an online diagnosis system called Monitor for arbitrary failures in distributed applications
• Diagnosis can be performed online with low latency
• Monitor’s probabilistic diagnosis outperforms Pinpoint by achieving higher accuracy for the same precision values
• Complex interaction of components in e-commerce system make accurate diagnosis challenging
• Clustering algorithms like the used in Pinpoint have high dependency on the complexity of the application
  – Need for many distinguishable transactions to achieve high accuracy
  – Transactions must touch almost all the components
• Future Work: How do we estimate diagnosis parameter values more accurately?

Backup Slides
Detectors and Injected Failures

- Internal and an external failure detectors are created to provide failure status of transactions to Pinpoint and Monitor.

- Faults are injected into Pet Store application components: Servlets and EJBs.

- Four different kinds of fault injection:
  1. Declared exceptions
  2. Undeclared exceptions
  3. Endless calls
  4. Null call

- The internal detector is more likely to detect the declared and undeclared exceptions, and the null calls.

- The external detector is more likely to detect the endless call.

Number of tightly coupled components for Pet Store

<table>
<thead>
<tr>
<th>Components where faults are injected</th>
<th># of tightly coupled components</th>
</tr>
</thead>
<tbody>
<tr>
<td>item.screen</td>
<td>3</td>
</tr>
<tr>
<td>enter_order_information.screen</td>
<td>2</td>
</tr>
<tr>
<td>order.do</td>
<td>3</td>
</tr>
<tr>
<td>ShoppingClientFacadeLocalEJB</td>
<td>5</td>
</tr>
<tr>
<td>CreditCardEJB</td>
<td>4</td>
</tr>
<tr>
<td>ContactInfoEJB</td>
<td>4</td>
</tr>
<tr>
<td>CatalogEJB</td>
<td>3</td>
</tr>
<tr>
<td>AsyncSenderEJB</td>
<td>2</td>
</tr>
<tr>
<td>AddressEJB</td>
<td>3</td>
</tr>
</tbody>
</table>