Self-Adaptive Software System Monitoring for Performance Anomaly Localization

2011/06/17

Jens Ehlers, André van Hoorn, Jan Waller, Wilhelm Hasselbring
Software Engineering Group
Christian-Albrechts-University Kiel
Application-level Monitoring

- Extensive infrastructure monitoring, application monitoring not widespread
- Reactive probe injection after performance drops

- Business Processes
  - Key performance indicators, e.g. process throughput, ...
- Services
  - SLO appliance, workload, ...
- Application
  - Response times, operational profile, ...
- Middleware Container
  - Thread/connection pool sizes, ...
- Virtual Machine
  - Heap size, ...
- Operating System
  - CPU/memory utilization, ...
- Hardware
  - Availability, reliability, ...

Business/service monitoring

Application monitoring

Infrastructure monitoring
Kieker Monitoring Framework

- Flexible, integrated monitoring framework
  - Extension points for custom probes, records, logs, analysis plugins
  - Provides a broad set of analysis plugins
  - Evaluated in industry case studies, low (linear) overhead
Self-Adaptive Software System Monitoring for Performance Anomaly Localization

Kieker Analysis Tool

- Monitoring control panel based on Eclipse RCP/EMF
- Analysis/visualization plugins

- Dependency graphs
- Use case transition graphs

UML sequence diagrams
Inactive probes do not cause significant overhead.

Test system: Sun Blade X6270 with 2x Intel Xeon E5540, 8 cores at 2.53 GHz, 24 GB RAM, ZFS RAID, SunOS 5.1, Java HotSpot x86 Server VM 1.6
Self-Adaptive Monitoring

- Monitoring causes measurable overhead
  → Not all probes/measuring points are activated permanently

- Monitoring rules are continuously evaluated during runtime
  - Rule condition formulated via OCL [OMG10]
  - Runtime OCL expression evaluation on EMF models
OCL-based Monitoring Condition Example

- Select all interface and anomalous operations
  - OCL context element: instance of calling context tree (CCT) [ABL97]

- OCL expression

```ocl
class CallingContextTree {
    attribute root: CallingContext;
    attribute callingContexts: List<CallingContext>;
}

class CallingContext {
    attribute level: Integer;
    attribute anomalyScore: Double;
    association callingContexts: Collection<CallingContext>;
    operation op: Operation;
}

class Operation {
    attribute signature: String;
    attribute monitoringActivated: Boolean;
}
```

```ocl
class OCLMonitoringCondition {
    self.callingContexts->select(
        (parent.op.monitoringActivated and parent.anomalyScore > t)
        or level = 1) ->collect(op)
}
```
Anomaly Rating Procedure

1. **Response time forecast**
   Forecast expected response times based on historical observations.

2. **Anomalous behavior hypothesis test**

3. **Anomaly score calculation**

4. **Anomaly score aggregation and correlation**
Context Influences Response Times

- Contextual impact factors influence timing behavior of software services

- Example: Response times depend on CPU utilization and system workload
Response Time Forecast

- Response time observations form a univariate time series
- Assumption: Patterns observed in the past will remain in future
  - Allows short-term forecasts: \( x_1, x_2, ..., x_T \rightarrow \hat{x}_{T+1}, \hat{x}_{T+2}, ... \)
  - Stochastic process model has to fit to observed time series

- Implementation of different forecast models [Wei05]
  - Single exponential smoothing: \( \hat{x}_{t+1} = \beta x_t + (1 - \beta)\hat{x}_t \)
  - Holter-Winters-Smoothing
  - ARIMA-models, e.g. ARIMA(1,1,1): \( \hat{x}_{t+1} = x_t + \varphi(x_t - x_{t-1}) + \theta(x_t - \hat{x}_t) \)
Anomalous Rating Procedure

1. **Response time forecast**
   Forecast expected response times based on historical observations.

2. **Anomalous behavior hypothesis test**
   Test if a sample of newly observed response times is to be rated as normal or anomalous related to the expected value from 1.

3. **Anomaly score calculation**

4. **Anomaly score aggregation and correlation**
Anomalous Behavior Hypothesis Test

- Distribution of response times is unknown
  - Not possible to extract contextual impact factors completely
- Leverage central limit theorem
  - Not every operation execution has to be classified
  - Instead: Testing a sample of operation executions
Anomalous behavior hypothesis test
Test if a sample of newly observed response times is to be rated as normal or anomalous related to the expected value from 1.

3. **Anomaly score calculation**
Update anomaly scores for each operation representing its current anomaly degree based on the sequent rating of response times samples in 2.

4. **Anomaly score aggregation and correlation**
Aggregate and correlate anomaly scores from 3. to higher levels of abstraction, e.g. component-level anomaly scores. [MRvH+09]
Self-Adaptive Software System Monitoring for Performance Anomaly Localization

Anomaly Localization Evaluation
Self-Adaptive Software System Monitoring for Performance Anomaly Localization

Anomaly Localization Evaluation
Conclusion

▪ Motivation for application-level software system monitoring

▪ Architecture and application of the **Kieker** monitoring framework

▪ **Self-adaptive monitoring approach** for performance anomaly localization
References