Optimizing Elastic IoT Application Deployments

Zach Leidall
Overview

- DIANE
- MADCAT
- LEONORE
- Optimizations
- Evaluation
- My Thoughts
- Discussion Questions
Overview

**Problem:** IoT application deployment systems are currently static in nature and therefore unable to adapt/scale to changes in program needs or execution environments.

**Solution:** DIANE, a system which monitors edge deployments and performs optimizations based on a set of rules defined by the programmer.
DIANE

Built on ‘IoT Gateway’ premise.

Framework for allocating Edge resources to be used in IoT applications.

Based in the cloud.

Provisions edge compute resources.

Takes in an application and its MADCAT files.

Acts as a ‘master’ for managing the elastic infrastructure.
MADCAT

A declarative, JSON-like language for specifying application requirements.

Introduced by Michael Vogler in a previous paper concerning elastic cloud provisioning.

MADCAT: A Methodology for Architecture and Deployment of Cloud Application Topologies
Technical Units

Contains details required to run a particle component.

- Execution instructions
- Required Libraries
- Other TUs that it depends on
- DU required to run
- Executable files
Deployment Units

Contain information about the system/hardware that a TU requires to run. Generated based on TUs.

- CPU/Memory requirements
- Operating System details
- Special hardware capabilities
MADCAT Examples

Listing 1: Technical Unit
```
{
    "@context": "http://madcat.dsg.tuwien.ac.at/",
    "@type": "TechnicalUnit",
    "name": "BMS/Unit",
    "artifact-uri": "...",
    "language": "java",
    "build": {
        "assembly": {"file": "unit.jar"},
        "steps": [{"step": 1, "tool": "maven", "cmd": "mvn clean install"}]
    },
    "execute": [{"step": 1, "tool": "java", "cmd": "java -jar @build.assembly.file"}],
    "configuration": [{"key": "broker.url", "value": "@MGT.broker.url"}]
}
```

Listing 2: Deployment Unit
```
{
    "@context": "http://madcat.dsg.tuwien.ac.at/",
    "@type": "DeploymentUnit",
    "name": "BMS/Unit",
    "technicalUnits": [{"name": "BMS/Unit"}],
    "constraints": [{
        "hardware": [{"type": ".", "os": ".", "capabilities": [{"name": "JRE", "version": "1.7"}], "memory": "."}],
        "software": [{"replication": [{"min": "all"}]}
    },
    "steps": [...]
}```
Creating a Deployment Instance

1. Application developer creates a series of TUs.
2. DIANE generates the DUs required to support the TUs.
3. DIANE builds the dependency graph between the TU/DUs.
4. DIANE uses a constraint handler to generate a list of nodes which satisfy the constraints.
5. DIANE deploys the nodes via LEONORE, attaching monitors to control if/when optimizations occur. N+1 nodes are allocated.
Optimized Version

- More ‘hot’ nodes are pre-provisioned. This means the application components are deployed but not started.
- Applications have monitors attached to them which report back to the DIANE master who deployed it.
- DIANE monitors the application to see if required conditions are violated. These conditions can be specified in an Optimization Unit.
- If the conditions are violated, DIANE redeployes the application.
LEONORE

Result of previous work by Michael Vogler.

Controls the provisioning of IoT devices.

Runs on edge nodes

DIANE queries LEONORE to obtain the set of potential devices to deploy on.

**LEONORE -- Large-Scale Provisioning of Resource-Constrained IoT Deployments**
Evaluation Part 1

The ‘evolved’ architecture takes twice as long as the traditional one to deploy.

The authors explain this by adding that the number of ‘artefacts’ being deployed in the evolved is almost twice as much as in the traditional example.

Why?
Evaluation Part 2

The CPU and memory utilization of the modified version is initially higher than that of the traditional method.

Why?
Evaluation Part 3

Significantly reduced bandwidth due to the existence of program logic on the IoT devices themselves.
Evaluation Part 4

The execution time in the evolved version is lower for their test applications.
Evaluation Part 5: Black vs White Box
Evaluation Part 5: Black vs White Box
Positives

- Layered, modular approach
- Clearly provides a scalable and elastic architecture
- Thorough Evaluation
- Reduce the bandwidth to 17%
- Fault tolerance!
Negatives

- Overestimation of IoT processing resources.
- Deployment time and resource allocation is larger than the non-elastic method.
- Requires more effort on the part of the developer to make their application compatible.
- Comparisons only to their baseline.
Unresolved Questions

- Can devices be allocated to multiple applications?
- Why can’t edge devices manage their own elasticity rules?
- Does not say whether state is preserved when redeploying an optimized topology.
- No mention of code changes.
Discussion Questions

Are you convinced the optimizations offer a significant improvement?
Could you see this being used in a practical setting?
Blog Question

Can you think of any additional optimizations?
Other Questions?

???
References

Optimizing Elastic IoT Application Deployments

**Michael Vogler**, Johannes M. Schleicher, Christian Inzinger, and Schahram Dustdar, "Fellow, IEEE

LEONORE – Large-Scale Provisioning of Resource-Constrained IoT Deployments

**Michael Vogler**, Johannes M. Schleicher, Christian Inzinger, Stefan Nastic, Sanjin Sehic and Schahram Dustdar "Distributed Systems Group, Vienna University of Technology, 1040 Vienna, Austria

MADCAT A Methodology for Architecture and Deployment of Cloud Application Topologies

Christian Inzinger, Stefan Nastic, Sanjin Sehic, **Michael Vögler**, Fei Li, and Schahram Dustdar Distributed Systems Group, Vienna University of Technology