Adapting applications to exploit virtualization management knowledge

Vitalic A. Danciu and Alexander Knapp

DMTF SVM 2013
Applications running on virtualized infrastructure suffer!

1. Example of “suffering”, by experiment
2. How to adapt applications’ behaviour
3. Relation between application and the management system
Impact of virtualization on applications

An Inter-VM communication experiment

co-located

distributed

choked

App. component
VM and guest OS
Hypervisor
Physical network
Impact of virtualization on applications

Transmission time for $10^7$ bytes UDP payload (sender’s view)

Performance seems equal; how can this be?
Impact of virtualization on applications

Segment drop rate (receiver’s view)

Deployment setup can change at any time: how can applications adapt?
Observation: Deployment changes quickly, but application behaviour does not.

- Application operates from static-world assumptions.
- Virtualization masks deployment state from application

⇒ Detrimental effects in some of the states

Scope of the problem (network throughput is but an example!)

- locality: communication metrics (throughput, delay, faults, ...)
- resources: CPU capacity, RAM, ...
- context: security, hardware capabilities, ...

How to render application software virtualization-aware?
Problem analysis
Solution dimensions

Select one point in this solution space!
Vitalian A. Danciu and Alexander Knapp

Problem analysis
Source of information, point of decision

knowledge of the application’s needs
fine-grained, procedure-level

knowledge of the environment’s state
application
local, limited
guest OS
coarse, capacity-level
global, (ideally) comprehensive
hypervisor
management system

Choices
- Management system to provide environmental information/guidance
- Application to decide on it
Problem analysis

Point and method of change

Choices

- Modification of application code (source or binary)
- Programmatic, machine-supported modification
Modifying applications

Code example: output to a UDP socket

```c
int write_udp(const char* targetip,
              unsigned int port,
              long count) {
    int transmit_socket = socket(AF_INET,
                                 SOCK_DGRAM,
                                 IPPROTO_UDP);
    connect(transmit_socket,
            (struct sockaddr*)&si_other,
            sizeof(si_other));
    int c = 0;
    while (c++ < count)
        write(transmit_socket,
              (void*)chunk, (size_t)sbuf);
}
Adapting applications to exploit virtualization management knowledge

Aspect-oriented programming (AoP) at a glance

“Glorified string substitution with knowledge of the programming language”

- **Aspect**: “a cross-cutting concern”
- **Aspect language**
  - matching/scoping of language constructs (classes, methods, . . .)
  - access to program structures
  - manipulation of parameters, return values, . . .
- **Pointcut expressions**: where to modify
- **Advice expression**: how to modify
- **Weaving**: integration of Aspect code
Modifying applications

Example: a buffering aspect (for both sockets and files)

```java
1 aspect Buffering {
2    pointcut openpc() = "% ....: open(...)"; ..........Intercept the function calls, 
3    pointcut socketpc() = "% ....: socket(...)";
4    advice call(openpc() || socketpc()) : around() {
5        tjp->proceed(); ..........Allow the call to execute 
6        int myfh = *((int*)tjp->result()); ..........and to yield a file handle,
7        writebuf[myfh] = (char*)malloc(bufsize); ..........Associate the file handle with a buffer.
8 }
9
10   pointcut writepc() = "% ....: write(...)"; ..........Intercept the write() calls.
11 advice call(writepc()) : around() {
12        int fd = *((int*)tjp->arg(0));
13        const void* buf = *((const void**)tjp->arg(1));
14        unsigned int count = *((unsigned int*)tjp->arg(2));
15        int myfh = _getmyfh(fd);
16        if ((buffill[myfh] + count) < bufsize) {
17            memcpy(writebuf[myfh] + buffill[myfh], buf, count);
18            buffill[myfh] += count;
19        }
20        else { ..........When full, flush buffered data,
21            write(fd, writebuf[myfh], buffill[myfh]);
22            buffill[myfh] = 0;
23            tjp->proceed(); ..........and allow the current write() to execute.
24        }
25        *((int*)tjp->result()) = count; ..........Always return the expected value.
26    }
27 }
```
Modifying applications

Behaviour of original vs. woven code

![Graph showing wallclock time (s) vs. chunk size (bytes) and buffer size (bytes) for both with and without aspects. The graph compares the performance of the original code versus the woven code with aspects.]
Outlook

Supplying management information to applications
Your software runs on virtualized infrastructure—and it suffers!

- Environment has changed; application code has not.
- Need to adapt application code, but
  - not manually (too large code base)
  - not centrally (different applications have different needs)
  - not to be self-adaptive (selfish adaptation obviates management goals)

Our AoP approach works, however it has limitations

- Applicability determined by code quality → quality metrics?
- Conflicting aspects → balancing? aspect “patterns”?

Management is capable to supply global information, however

- it may not wish to (public XaaS scenarios) → discovery, gossip?
- which information is relevant to ask? → situation/cause/effect? formalism?