CS428: Software Engineering II

Performance

Slides are modified based on Cosmin Radoi’s slides for Spring 2014
What is “performance”? 

The word performance in computer performance means the same thing that performance means in other contexts, that is, it means “How well is the computer doing the work it is supposed to do?”

Arnold Allen

- High speed, or high throughput
- Good responsiveness (short response time)
- Low energy consumption
- Good usability, good ROI, ...
What is “performance”?  

*The word performance in computer performance means the same thing that performance means in other contexts, that is, it means ”How well is the computer doing the work it is supposed to do?”*  

Arnold Allen  

Good performance =
What is “performance”?

The word performance in computer performance means the same thing that performance means in other contexts, that is, it means ”How well is the computer doing the work it is supposed to do?”

Arnold Allen

Good performance =

- high speed, or high throughput
What is “performance”?  

The word performance in computer performance means the same thing that performance means in other contexts, that is, it means “How well is the computer doing the work it is supposed to do?”

Arnold Allen

Good performance =

- high speed, or high throughput
- good responsiveness (short response time)
What is “performance”?

The word performance in computer performance means the same thing that performance means in other contexts, that is, it means "How well is the computer doing the work it is supposed to do?"

Arnold Allen

Good performance =

- high speed, or high throughput
- good responsiveness (short response time)
- low energy consumption
What is “performance”? 

The word performance in computer performance means the same thing that performance means in other contexts, that is, it means “How well is the computer doing the work it is supposed to do?”

Arnold Allen

Good performance =

- high speed, or high throughput
- good responsiveness (short response time)
- low energy consumption
- good usability, good ROI, ...
Why optimize for performance?
Why optimize for performance?

Because...
Why optimize for performance?

Because...

PERFORMANCE

SUCH GOOD
Why optimize for performance?

- Clear distinctions
  - Soft performance requirement
  - Hard performance requirement

Generally, when to optimize for performance?
- Do NOT optimize during development
- When a performance need is clearly identified, e.g.,
  - You do alpha testing, and notice regular lags
  - You receive bug reports
  - Particular UI/integration tests are slow
  - A/B testing and the results don't add up
## Why optimize for performance?

### Clear distinctions

- Generally, when to optimize for performance?
  - Do **not** optimize during development
  - When a performance need is clearly identified, e.g.,
    - You do alpha testing, and notice regular lags
    - You receive bug reports
    - Particular UI/integration tests are slow
    - A/B testing and the results don’t add up
    - ...
## Why optimize for performance?

### Clear distinctions

- *soft* performance requirement

**Generally, when to optimize for performance?**

- do NOT optimize during development
- when a performance need is clearly identified, e.g.,
  - you do alpha testing, and notice regular lags
  - you receive bug reports
  - particular UI/integration tests are slow
  - A/B testing and the results don’t add up
  - ...
Why optimize for performance?

Clear distinctions

- *soft* performance requirement
- *hard* performance requirement

Generally, when to optimize for performance?

- do NOT optimize during development
- when a performance need is clearly identified, e.g.,
  - you do alpha testing, and notice regular lags
  - you receive bug reports
  - particular UI/integration tests are slow
  - A/B testing and the results don't add up

...
Why optimize for performance?

Clear distinctions

- *soft* performance requirement
- *hard* performance requirement
Why optimize for performance?

Clear distinctions

- *soft* performance requirement
- *hard* performance requirement

Generally, *when* to optimize for performance?

- do NOT optimize during development
Why optimize for performance?

Clear distinctions

- *soft* performance requirement
- *hard* performance requirement

Generally, *when* to optimize for performance?

- do NOT optimize during development
- when a performance need is clearly identified, e.g.,
  - you do alpha testing, and notice regular lags
  - you receive bug reports
  - particular UI/integration tests are slow
  - A/B testing and the results don’t add up
  - ...

### Throughput (Velocity)

Amount of data processed (computed, transmitted, etc.) in a unit of time.
Some definitions

Throughput (Velocity)

Amount of data processed (computed, transmitted, etc.) in a unit of time.

- database transactions per second
- web pages served per second
- data entries computed per second
Response time

Amount of time in which a result is produced.
Some definitions

Response time

Amount of time in which a result is produced.
- amount of time to serve a web page
- amount of time to clear a credit card transaction
- amount of time to get web service response
- ....
Web response time

- 0.1s - feels instantaneous
- 1s - no disturbance in the flow of work
- 2s - average waiting time, attention span
- 10s - keep on task

- Nah, Fiona Fui-Hoon. ”A study on tolerable waiting time: how long are web users willing to wait?.” Behaviour & Information Technology 23.3 (2004): 153-163.
progress bars are cheaper than performance optimization

Pareto principle

- 80% of the effects come from 20% of the causes
Group Discussion

☐ Get into a group of 2 (or 3 students), share with each other one or two examples of performance faults that you experienced or observed

◊ What it was about
◊ How you found the performance failure
◊ How you diagnosed the failure and hunted out the root cause (fault)
◊ How you fixed the fault

☐ Pick random/sample groups to share their discussion outcome
Pareto principle

- 80% of the effects come from 20% of the causes
- for programs, 80% of the time is spent in 20% of the program
Amdahl’s Law

The improvement in performance is limited by the fraction of the overall time used by the optimized part of the system.

E.g., querying the database takes 100ms out of the 300ms required to serve a webpage. What speedup would you get with a perfect database (which takes 0ms to query)?
Maybe it is obvious...

Reducing the lines of code in a high-level language improves the speed or size of the resulting machine code.

A fast program is just as important as a correct one.

“Old Wives’ Tales” identified in Code Complete.
Maybe it is obvious...

- Reducing the lines of code in a high-level language improves the speed or size of the resulting machine code.
Maybe it is obvious...

- Reducing the lines of code in a high-level language improves the speed or size of the resulting machine code.
- A fast program is just as important as a correct one.
Reducing the lines of code in a high-level language improves the speed or size of the resulting machine code.

A fast program is just as important as a correct one.

“Old Wives’ Tales” identified in Code Complete
Sources of inefficiency

- multilayer architectures
- high-level languages
- I/O
- bad design decisions
- inappropriate use of data structures
- expensive algorithms
L1 cache reference: 0.5 ns
Branch mispredict: 5 ns
L2 cache reference: 7 ns
Mutex lock/unlock: 100 ns (25)
Main memory reference: 100 ns
Compress 1K bytes with Zippy: 10,000 ns (3,000)
Send 2K bytes over 1 Gbps network: 20,000 ns
Read 1 MB sequentially from memory: 250,000 ns
Round trip within same datacenter: 500,000 ns
Disk seek: 10,000,000 ns
Read 1 MB sequentially from network: 10,000,000 ns
Read 1 MB sequentially from disk: 30,000,000 ns (20,000,000)
Send packet CA->Netherlands->CA: 150,000,000 ns
Profiling

- starts with or attaches to your running program
- gives detailed information about its resource utilization
- demo VisualVM ...
Low-level tuning

- order tests by frequency
- choose better logic structures
- table lookup instead of boolean logic
- loops
  - unswitching, fusion (jamming), unrolling, reorder loop nest
  - minimize work inside a loop
  - terminate loops early
- use integers instead of floating point
- algebraic identities
- strength reduction
- recode in low-level language
Why NOT do low-level tuning?

- many optimizations are platform dependent
- good compilers are much better than average humans
- interpreted languages also benefit - JITs
Caching

▶ very powerful
▶ for the functional programming inclined, same as laziness
▶ can also be done at compile time - precompute values
Caching

- very powerful
- for the functional programming inclined, same as laziness
- can also be done at compile time - precompute values
Caching example

```java
Output compute(Input input) {
    ... 
}
```

```java
private Output eagerCompute(Input input) {
    ... 
}
```

```java
Output compute(Input input) {
    Output output = precomputed.get(input);
    if (output != null) {
        return output;
    } else {
        output = eagerCompute(input);
        precomputed.put(input, output);
        return output;
    }
}
```

Caching example

Output compute(Input input) {
    ...
}

Map<Output, Input> precomputed = new Map<Input, Output>()
private Output eagerCompute(Input input) {
    ...
}
Output compute(Input input) {
    Output output = precomputed.get(input);
    if (output != null) {
        return output;
    } else {
        output = eagerCompute(input);
        precomputed.put(input, output);
        return output;
    }
}

↓

Map<> precomputed = new Map<Input, Output>()
private Output eagerCompute(Input input) {
    ...
}
Output compute(Input input) {
    Output output = precomputed.get(input);
    if (output != null) {
        return output;
    } else {
        output = eagerCompute(input);
        precomputed.put(input, output);
        return output;
    }
}
CheckRole (temporal)

Cost: 16% of 30% instruction-count overhead
Path:
  preInvoke
calls performAuthorization
calls ejbCheckAuthorization
calls checkAccess

Observation: decision is (almost) deterministic
CheckRole, optimized with cache

```
performAuthorization methodInfo inCreds beanCookie

tbl = getTable(methodInfo)
if (tbl.hits(inCreds, beanCookie))
  return tbl.value(inCreds, beanCookie)
else
  creds = compute creds the slow way
  tbl.add(inCreds, beanCookie, creds)
return creds
```

outCreds
GetCred (spatial)

Cost: 13% of 30% instruction-count overhead

Path:
preInvoke
  calls doPrivileged
calls run
calls get_credentials

```
preInvoke
  doPrivileged // Expensive!
    creds = get_credentials()
    if (!isGrantedAnyRole(roles, creds))
      Scream!

// In another class, far far away
public get_credentials()
  // Expensive!
  stack = getAccessContext()
  checkPermission(stack, canReadCreds)
  return creds
```
GetCred, optimized by specializing

```java
private static boolean ok = false;
preInvoke
    if (ok) creds = get_credentialsQuickly()
else doPrivileged  // Expensive!
    creds = get_credentials()
    if (!isGrantedAnyRole(roles, creds))
        Scream!

// In another class, far far away
public get_credentials()
    // Expensive!
    stack = getAccessContext()
    checkPermission(stack, canReadCreds)
    ok = true
    return creds
public get_credentialsQuickly()
    return creds
```

- Wide application
- Proposed for IBM JIT
DBReuse

Cost: 10% of 30% instruction-count overhead

Paths:
- getConnection
  - calls <5 methods>
  - calls doPrivileged
  - calls <2 methods>
  - calls Subject.equals

- getConnection
  - calls <2 methods>
  - calls getSubject
calls doPrivileged

- getConnection
  - calls <7 methods>
  - calls Subject.hashCode

Optimization:
Cache results of equals(), hashCode(), getSubject()
**Reflection**

Cost: 25% of 30% instruction-count overhead

Path:

```
CacheableCommandImpl.execute
calls CacheableCommandImpl.setOutputProperties // uses reflection!
calls doPrivileged
calls run
calls checkMemberAccess
```

Optimization: avoid reflection by overriding `setOutputProperties`
Throughput improvements, by optimization

Normalized throughput

- No
- J2SE
- Global
- Full

Legend:
- Original
- All
- CheckRole
- DBReuse
- Reflection
- GetCred

Finding and Removing Performance Bottlenecks in Large Systems, December 2, 2004. 27