8/29/2013 - “THE”

Multiprogramming vs batch

Hierarchy

Proof of correctness

Virtual – segments vs pages: segment here is fixed size (or multiple pages), particular address.

Virtual cpu

Semaphores

Processes

Testing

Software development

No multi task

No security

No prority

Higher level can use lower level, but inverse is not possible.

In real world the hierarchy is hard to achieve.

Network in: 比如 level3: 好处可以用其以下的，坏处其以下不可用network. 若不是用hierarchy，则可以用module，every dependent on other，但可以互相使用。

4: applications

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3:devices

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2:console : act like every process has its own console

(If exchange level 1 and 2, might let not so important message of console occupy in the core page)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1： segment: physical memory

\_\_\_\_\_\_\_\_\_

0: Cpu process (resource management, which process run at cpu, abstraction of process, real-time clock)

9/3/2013 – Multics

Large virtual memory

Dynamic linking – sharing of code

Virtual addresss

Segments:

Data & code for each process, variable size, access list, can have >= 1 symbolic name e.g. “/bin/data”, single level store(no distinction between file and segments), distinct, grow and shrink

Processes:

Composed of segments, each process has its own single address space

Stack & stack pointer

Weakness:

Overhead (e.g. caused by dynamic linking)

Fragmentation

Paging algorithm

**Segments: composed by pages, has a page table for the pages of the segment.**

9/5/2013 – Hints

Simplicity

Speed

Don’t hide the power of underlying things

Procedure arguments

Leave it to the client

.VAX e.g. CISC

.i432 (Intel 432)

Keep interfaces stable

.backwards compatible

Plan on rewriting

**Separate normal & worst cases: ! Very important**

Make things static: static allocation is easier than dynamic allocation

**Caching**

– keep the answer

**Hints**

– hint the answer(this may be wrong), if want exactly right it is too expensive thus we use hints.

Hints is kind of speed up.

Hints need cheap verification

Better mostly right

**Watch constant factors**

.Brute force

.simple

**Do work in the background:** most files do not exist very long, may wait not to write them into disks.

**Batch**

Avoid mistakes

Shed load:

drop out of line if it works bad, if might give other users get the computer works better, don’t do a job on everything

**end-to-end:**

.application-level checks for correctness

.lower-level checks for performance

**Logging**:

.atomic or restartable actions

Dynamic translation

JIT compiling

VMWare

09/10/2013

LFS

Keep it simple

Logs

Caching

Crash recovery

Work in the background

Batching

Separate normal from worst cast

Fast indexing-hint

How to speed up disk performance?

.do not use the disk.

Don't use the disks

Caching

Use the disk efficiently

.scheduling algorithm

Lag data out on disk intelligently

FFS-spatial locality

e.g. all blocks of a disk at the time the file

a.. files in the same directory is written

LFS-temporal locality

.blocks written together are read together written

technology trends:

.CPUs are getting faster than that of disks

.big memories

.disk arrays – small writes defects the performance of disk arrays

.SCSI- variable disk gemetric

LFS – treat the disk like a tape,少读少扫描，写快读慢

* + 1. problems:

1. what happens when the disk fills up?

Solution:

* + - * 1. buy a new disk – Plan 9 file system

2.how do I read from the disk.

.inode

.inode map

.log wrap

.garbage collection: log clear

.break disk into fixed-size segments

09/12/13

.cleaner – creates new empty segments

.copy live data out of partially-full segments

.live data – segment summary block

Refer to the inode

.cleaning policy – greedy algorithm: lead to poor performance. because might somebody already will to clean the segment(this seg is a hot seg), so to clean it now is kind of a waste.

.need to take the age of the data into account.

.checkpoint to speed up crash recovery.

.check points at a known location.

.**Flash memory: no seeks cost. Cannot write a block until it is erased. When erase must erase > 1 sector. Erase times is limited.**

**Use LFS for flash memory’s Benefit:**

**. tie a number of sectors to a segment in LFS, flash memory can only be written with several sectors together, not individually.**

**.flash memory has a drawback each sector can only be written for limited times, LFS enable to write wherever sector we would love to write. Moreover, other file system asks for writing in certain fixed locations thus conflict with the flash memory wear level.**

**Synthesis**

Specialization – e.g. 4 different types of queues for particular situation.

2 different context switch

Code synthesis – runtime code generation to specialize a particular scenario.

Every thread has special quantum to save its states.

**Figure 3 explains context switch:**

缺点：multiple copies of context switch code.

Procedure chaining

09/17/13

1.Critical section:

Roll backword 方法:

Count++;

实现过程：

Do

Old =count;

New = old + 1;

While(cas(count, old, new) == false) // cas (comparable swap): compare to see whether 运算时有其他thread进入critical section.

Roll forward 方法: 不管其他process想进入critical section, 等本process运行完critical section再说。

**Scheduler Activations**

**Exam question:**

1.user threads vs kernel threads

Kernel threads

.threads are implemented by the kernel

.system calls for thread operation.

.Goodside: blocking operations

Can Schedule on multiple cores

Downside: overhead, not feasible to create many kernel level threads

.the number of kernel threads = # of kernel threads blocked + # of processors

User-level threads

.implemented by the application. (context switching ,control box…etc)

.function calls for thread operations (function calls much faster than system calls)

.Downside: OS doesn’t know about them,

e.g .**blocking operations**: for example read are problems.

e.g. can’t schedule threads on multiple cores.

Note: about blocking operations, In user-level thread, if one user-level thread blocked, all the other threads would also blocked. However, in kernel-level thread, if one kernel level thread blocked, the system could choose other available kernel-level thread to run. Thus, kernel –level thread would not have blocking operation problem.

2.Paper P66 图: application provide to OS

Add more processors()

This processor is idle()

3.Paper P62图: Kernel provide to application

Add this processor(): execute a runnable user-level thread

Processor has been preempted(): the kernel has decide application has too many processors had took processor away

Scheduler activation has blocked(): the thread running now is blocked

Scheduler activations has unblocked: put thread in the run queue waiting for running

4.Paper P63图 page fault 怎么办， the scheduler activations work properly even when a preemption or a page fault occurs in the user-level thread manager when no user-level thread is running.?????

09/19/2013

1.Paper P47 图： Binding

2.Paper P53图: if >1 call packets, need ACK response to consent the sent of packet successfully.

3.Paper P51 图片: if server machine fails, the client keep sending message ask for the server to do things, there is a possibility that the server do the procedure for several times which is not desirable like给一个账号加100元，而不是set 为100元。此处就有问题。If the client the kill the procedure, the process just stopped without accomplishment.

09/24/2013

LRPC:

1.RPC good point:

.if server or client go down, the other side still not down.

.the privacy and isolation between client and server can be achieved.

10/1/2013

Speculative execution:

10/3/2013

Speculative execution:

Benefit:

1. hide latency
2. improve server throughput

drawback:

1. assume we know the answer
2. need speculative processes
3. speculative objects, e.g. files
4. need a way to undo the speculation

d.1 things that can’t be undo: e.g. external IO – console, network messages

Note: have to resolve speculation

10/8/2013

Zebra

1.

10/10/2103

10/17/2013

Azure:

1.与GFS之不同:

Azure:

不用leases,

strong consistency;

all three replicas identical（用sealing方法: seal no modify to the extend, last extent marked unsealed for future store data）, 而GFS会有append duplicated

client just report the stream manager that the write fail, then stream manager ask client to try new extents. Truncate the extent, if not consistency. Move on a new extent seal the old extent.

Once extent sealed it cannot be modified, 可以做以下改进take for parity, stored the extent across space to reduce overhead.

2.partition layer:

partition manager: PM is just choosen among the partition server as a PM. First partition server who gets the lock to be PM(lock service 通知其他人). 缺点: 不知拿到lock的是不是malicious. There is Lease for PM . PM decides which partition server to be mapped according the state written by the client in table. Pm可以把overload的data从一个partitnin server range move到另一个，因为各个PS共享同一个stream layer所以此过程easy.

3.persisten data structure

commit log: check point the states,

1.see if th range is in commit log or not, if not check the checkpoint.

2.metadata strream

3.memory table: try to avoid the disk I/O,

4.bloom filter: give quickly way whehter sth is exist in disk or not. 缺点只能给might exist 不会说一定exist.

4.jounaling:

provides better avaerage performance, slow thingks down in normal case, but avoid high variance.

5.append-only system:

6. mulitple data abstractions from a single stack: why not separte blob, queue, tables in different machine, because 会resource underutilized.

7.end-to-end checksum:

10/22/2013 Needle

millinion photos/second

long tail

relative small

goals:

high throuhput & low latency

fault-tolerant

cost-effective

simple

L = 1 disk access / photo

many photos in a file instead one photon per file.

CDN: content distribution network

Needle: photo need to be displayed, just append the needle once want to store a new photo.

Garbage prob:

index in memory: to specify the photo in disk.

Possible: needles deleted after last checkpoint, com back go an read needle in disk to see the flag in needle to get the consistency.

Needle: cookie option:

protect people from iterrate the all photos by providing a large random number in URL.

Garbage collecion: for needle's flag saying delete.

Copy live needle to new file.

10/25/2013

把切成很多份，看checksum

prob:

1. hash: solu:用SHA-1, 把any size, any data变成 160 bits的number,

2. mis-alignment(开头多了几个字母，没有对齐): Rsync想解决： divide the data into 4 charaters chunks, compute all possible 4 character chunk with different start point.

本文solu: everytime see a letter “D” - 因为字母D出现频率合适，不高也不低， it is a new block.（base on content of the file）

need database: to obtain hash value, database only hints for not consistency, just keeping reference

writes: from a client to the server, to the server all the client got, the server tell what he has.

Close\_open\_semantics: write first to temporray file once done then do actually write; Reason: maintain old file to hash when need.

Prob: if server crash, on client side still old version, on server side keep temporay file.

Database: only has hints, if wrong just need extra RPC to get the right

do want block to be too small: if too small, computing consumtion, and size consumption. Solu: set minumu bk size, max bk size, if outer just ignore that bk.

Lesase mechanism: open a file get a lease on the file, som promise ont the file whehthrer is good, so do not need to tell the server whether it is changed or not.

10/29/2013 Disco

1.Virtual machine: very old idea.

2.spechar

3. Disco:

main idea: what to do with mulit process machine.

Motivation: mulitprocesses machine are just around the corner

need a muliprocessor OS

need to add locking to your kernel if multiprocessors

**4.virtual machine monitor:**

.advantage: much simpler software than OS.

5.use districuted system technology: e.g. TCP/IP & NFS

e.g. share files across different cores: use NFS to do that.

**6. P12 figure 2**

each has the replica in machine page for fast access, only replica read-only pages.

7. **P14 figure 4**

just want one exist, share in machine memory.

8.**P16 figure 5**

just like change the pointers. To get the copy to work.

9.how much going slow you down to have vm and vmm? The overhead problem

10/31/2013 Xen

1. instead of OS tell VMM what to do with instruction, tell VMM what to do with hypercalls.

2. use ring

3. P5 firgure 1:

virtual blocks device: virtual disk map to real disk,

**domain0** control interface: control operation on other domains. If it crashes,

Xen cannot run, then guest OS cannot run. (because of the device drivers).

4. P6 figure 2: ring buffer, 4 pointers for data structure transfer between Xen and guest OSes. Problem: do synchronization for 4 pointers, guest OS and Xen share ring buffer.

5. page tables: OS cannot manipulate page tables directly, Xen will lokk up at what to do to see whhether OS have the persimission.

Writes-hupervisor(hypercall): VMM check this then make the change.

Read page tables can do read directly.

Map Xen to upper layer OS, **Disco** not have this prob.

Cache results to page table lookup, **translation look aside buffer**(TLB). Next time do page table look up also check TLB, if exisit then save time. Else ask OS what should I do send an exception, let OS deal with it, do not need us complicate x86 page tables look up.

Tag TLB: entries for mulitple process, tag for which process check TLB there.

6. bulloon driver:

when create a domain from Xen, Xen set guest OS how much memory it has. Add bulloon driver to guest OS, it is working with Xen. Xen call bulloon driver to say this OS should use less memory, guest OS reduce the physical memory to tell bullon dirver, then bullon dirver tell Xen. Then may use it for other guest OS. Note: memory 的增大减小不能超过set guest OS 设置的空间大小，只能 在之内变化。若guest Os实在没有memory可以减小，Xen又必须要，则Xen stops that whole guest OS.

Disk space 不能进行如此操作。只能对memory操作。

Vmware run in unix process.

11/5/2013 TTVM

MMIO read:MMIO(Memory mapping I/O)即[内存映射I/O](http://baike.baidu.com/view/1056577.htm)，它是PCI规范的一部分，I/O设备被放置在内存空间而不是I/O空间。从处理器的角度看，内存映射I/O后系统设备访问起来和内存一样。

OS hard to debug because non-deterministic, VM can make it determinisics, log non-determinitsct events and do it in deterministic way.

11/7/2013 Remus

11/12/2013 Grapevine

message servers – reliable delivery

registration services – authentication, resource discovery

assumption&goals

no assumption about message content

don't trust the clients, but trust the servers

tolerate Grapevine server failures – ultimate goal of distributed computing

good performance

decentralize administration

flexible

authentication

Registration database

key/value store : key == Rname, value = individual or machine or list(like list of Rnames)

two-lelvel hierarchy

name.registry

problem: how to find find registration server serve that registry? Use the registration database to store the info about registration database.

E.g. 有个CSC552 registry, ask CSC552.gv- give a list of all registration servers that store the CSC552 registry. Ask gv.gv – get a list of all resigtration server that has the gv registry.

Need some out of band mechanism a way to find registration server.

Solution 1: find initial RS – use broadcast saying anybody has the registration server. Assumption: the does not need to do too much broadcast. Problem of broadcast: overhead and security.

Solution 2: hard-code one RS address on the network

Solution 3: have a well-known name, but this requires to rely on the external name server.(e.g. DNS)

Scalability/availability

all registries are replication at 2+ RS.

Replicas may get out of date, no transaction support among the replicas. Pioneering: loosely consistency, eventually the system could reach some consistency. And the system might get partitioned in some way.

- eventually consistency: over some period of time get the consistency. The way to do that is propagate the changes between Registration Servers.