Sean Mertens  
Z1630956

CSCI 480

Graduate Presentation Summary

Report 1: Vazravirat Pasumarthi, Lottery and Stride Scheduling

In a multithreaded system, scheduling computations can be difficult. The nature of the computation in question is important – long-running computations (such as research simulations) require system resources to be regulated, and interactive computations (such as database access) require rapid resource allocation for immediate tasks. Traditional scheduling schemes are not sufficient for tasks such as these. Instead, we use Lottery and Stride scheduling. Both schemes utilize tickets, first-class objects that encapsulate resource rights. Tickets can be abstract (quantify resource rights independently of machine details), relative(fraction of a resource that they represent varies dynamically in proportion to the contention for that resource), or uniform(rights for heterogeneous resources can be homogeneously represented as tickets.). Lottery Scheduling is the first scheduling algorithm we will cover. It relies on probability, hence the name “Lottery Scheduling”. Processes are assigned a number of “tickets”. As more tickets are assigned to a process, its probability of being selected increases, much like purchasing raffle tickets to increase your chance of winning. The scheduler draws a random ticket to decide which process to select next. Stride scheduling is a deterministic allocation mechanism for implementing proportional share control system resources. Stride scheduling computes a representation of the time interval that clients must wait between successive allocations. This is the “stride”. A client with the smallest stride is scheduled most frequently. A client with half the stride of another will execute twice as quickly. Stride scheduling utilizes three state variables: Tickets - specifies the client’s resource allocation, relative to other clients. Stride- is inversely proportional to tickets, and represents the interval between selections, measured in passes. Pass - represents the virtual time index for the client’s next selection. Stride scheduling operates by selecting the client with the smallest pass, and its pass is advanced by its stride. While both these scheduling systems are inefficient in I/O intensive environments, they have their strengths. They offer dynamic control over resource consumption rates, eliminate starvation, and tickets encapsulate resource rights.

Report 2: Shaikh Ismail, Virtualization

Virtualization is used for running environments that may be different from the host, for example, a Linux system running on a Windows machine. The development of virtualization was motivated by the desire to not have to purchase expensive hardware. Virtualization enables a user to run multiple VMs (Virtual Machines) on one platform. You can run a network of virtual machines to simulate an actual network, and the underlying host is protected by the virtual machines. It is possible to take snapshots of the virtual machine at different times, create a template to standardize the virtual machine image, or utilize live migration. There are 3 types involved: Type 0 – (Hypervisor: hardware-assisted partitioning) Type 1 – (special-purpose application running on bare metal, does not require a host operating system) Type 2 – (App running on a host to provide an environment for virtual machines) There are many kinds of implementation. Paravirtualization is using virtual machines modified to run on hardware. Programming-environment virtual machines would be like the JVM, or Java Virtual Machine – they exist only in software. Finally, emulation is a type of virtual machine that is used to run gaming platforms on a computer, they “emulate” or imitate the hardware of a gaming system. In summary, commercial providers market virtual machines for everyone to lower operating costs, ease development and testing, and offer alternatives in the computing market. Remember, however, that this all comes at a cost – you are limited by your host. Emulating a powerful system with a weak one won’t magically give you more computing power!

Report 3: Bhargavi Manukonda, Memory Resource Management in VMWare Server

VMware ESX Server is a thin software layer, or kernel, designed to multiplex hardware resources efficiently among virtual machines running unmodified commodity operating systems. It manages system hardware for high-performance I/O. It runs on “bare metal”, in other words, it does not need a host operating system. The guest operating system needs to see a 0-based memory space. The machine address is the host hardware memory space. The physical address is the virtual machine’s memory space. ESX Server maintains a pmap data structure for each VM to translate these physical page numbers (PPNs) to machine page numbers (MPNs). A VM from which memory has been reclaimed should perform as if it had been configured with less memory. ESX Server uses “ballooning” to achieve such predictable performance by coaxing the guest OS into cooperating with it when possible. ESX Server controls a balloon module running within the guest, directing it to allocate guest pages and pin them in “physical” memory. The machine pages backing this memory can then be reclaimed by ESX Server. ESX must do memory reclamation without information from the virtual machine’s operating system. As far as memory sharing is concerned, running multiple operating systems in virtual machines may result in multiple copies of code running. It is possible to exploit data redundancy to reduce total memory usage. ESX utilizes page content to implement sharing, and ESX does not need to modify its host operating systems.

Report 4: Uma Saketh Gorrepati, Live Migration of Virtual Machines

Live migration is moving a running VM between different physical machines without disconnecting the client or application. Memory, storage, and network connectivity of the virtual machine are transferred from the original host machine to the destination . The challenges involved are minimizing downtime, minimizing total migration time, and not disrupting active services. The benefits of live migration are that it avoids residual dependencies, can transfer an in-memory state, and it allows separation of concern between users and operator of a data center or cluster. Migrating memory involves three phases: -a push phase (The source virtual machine runs while pages are pushed to a new destination. To ensure consistency, pages modified during this process must be re-sent.) –a stop and copy phase (The source VM is stopped, pages copied and a new virtual machine started) –a pull phase (New virtual machine executed, if “not yet copied” page is accessed it is pulled across the network to the new virtual machine from the source virtual machine) In conclusion, by integrating live operating system migration into the xen virtual machine, rapid movement of interactive workloads across a data center is possible. By using pre-copying and dynamically adapting network bandwidth total downtime can be reduced to

imperceptible levels, even for complex interactive services.

Report 5: Sriharsha Siram, Cloud Operating Systems

Cloud computing is a relatively new technology that has been popularized in today’s pop culture, mostly due to its potential. It is a new approach to deploying technology that allows clients to access computing resources at a certain time for a small price. Cloud computing, due to its complexity, requires a well-designed operating system to access and use. A well-designed cloud operating system should have: 1. Well defined interfaces that hide implementation details. 2. Core security services. 3. The ability to manage virtualization. 4. Management of Workloads to provide quality of service and performance. Applications In this environment should consist of well-defined services with well-designed interfaces, in order to remain interoperable and predictable. Data being manipulated should be treated as a separate set of services to that it can be more readily shared. The nature of cloud computing means that data can be shared across applications, customers, suppliers, and partners. The rate at which cloud computing can access different systems means that the operating system needs to provide a secure and consistent foundation. Without these deliberate design choices, the entire system may suffer. Well defined interfaces are necessary between software components in order to

achieve the desired level of flexibility and reliability. The operating system must be counted on to make sure the right connections are made between IT resources and the applications. This is, of course, a given in any operating system, but it Is even more critical when cloud computing is involved. An operating system in a cloud environment should also readily support security measures, and should be able to manage virtualized workloads. It should utilize a hypervisor – it brings order to a complex environment comprised of multiple images of various IT resources. A hypervisor enables a specific application to exist on many different systems without actually being physically copied onto each system. The Hypervisor relies on important capabilities within the underlying operating system to manage the interaction between applications and their interfaces so that they are well managed and protected. Finally, a cloud OS should optimize workloads and provide consistent service to a user. In conclusion, a cloud operating system should have maturity, scalability, manageability, and open source community support.

Report 6: Venkat Pasupuleti, Android Operating System Architecture

Android is a mobile operating system based on Linux developed and maintained by Google. It is fast, powerful, and easy to develop for. It is used mostly on mobile devices with touchscreens, although there are some exceptions. It supports almost all architectures that the original Linux kernel supports, making it a solid choice for coders with Linux experience. Android has over one billion active monthly users. It is the most used mobile operating system, and seventy-one percent of mobile developers use Android. It is clear that with the sudden interest in mobile development, Android will be the environment of choice for most mobile developers. Android is based on Linux, a Unix-like and mostly POSIX-compliant computer operating system. Linux uses free and open-source software development and distribution. It was originally developed as a free operating system for Intel x86-based personal computers. Linux is ported to more computer hardware platforms than any other operating system. It is also the leading operating system on servers and other big iron systems such as mainframe computers and supercomputers. 97% of the world's 500 fastest supercomputers run some variant of Linux.

Android at its core is based on the Linux kernel, which provides basic system functionality like process management, memory management, and device management. Next are the Android libraries, including the open-source web browser engine WebKit, and SQLite, a database. Next is the ART, or Android runtime. It features ahead-of-time (AOT) compilation - it compiles your application to native machine code when installed on a user’s device. It also features a new garbage collector and memory allocator. Above this is the application framework (which features higher-level services in the form of Java classes), until finally we reach the application level, which is where the user interacts. Android is easy to develop for, and is a must- use environment for anyone involved with mobile development.

Report 7: Pranav Nadigapu Suresh, The design and implementation of Log Structured file system

A filesystem is a piece of software which controls how data is read from (and written to) the secondary storage. As such, it is responsible for organizing files and directories. Recently in the computing world, we have seen much growth in the size of computers’ main memory. As a result of this, the cache memory size has also increased in order to speed up system operations. This large amount of cache memory is used as a write buffer. Filesystem design is dependent on the technology of hardware available, and the workload it is expected to handle. Filesystems normally rely on processors, disks, and main memory. As computer hardware advances, processor speeds are increasing significantly. Disks are becoming cheaper and larger, but they are not advancing in terms of transfer bandwidth or access time. Transfer bandwidth can be improved by utilizing parallel head disks/disk arrays, and access time is dependent on the actual motion of the disk. Main memory size has increased, which also has increased cache sizes. A larger cache means the disk has less work to do since read requests are handled by the cache. A larger cache can also be used as a write buffer to convert many small writes into a few large ones. Filesystems can be inefficient by having many accesses. Using a log based filesystem attempts to circumvent some issues that have plagued traditional filesystems. The fundamental idea is to cache the file system changes to a file log and write to disk in a single operation. In this case, the information written to disk includes all the information that is used to manage the file system. The file system then converts large small synchronous random writes to large asynchronous sequential writes. In order to save space, the log based filesystem utilizes threading (tries to write in the free extents available) and copying (copies the live data to contiguous blocks and free data blocks are collected to large groups). It tries to combine both the techniques by dividing the disk into large, fixed size extents called segments, which are written sequentially from beginning to end. A log based filesystem also has a built-in crash recovery, since recent disk operations reside at the end of the log. In conclusion, a log based filesystem handles reads through caching and writes by appending large segments to a log, and greatly increases disk performance.

Report 8: Abhijith Chinthamani, Security for Operating Systems

Any software that is expected to handle sensitive information or access a network must have built-in security to prevent unauthorized access. An operating system, with all of its functionality, must therefore have built-in security features to protect the data integrity of its user. Many threats exist, such as unauthorized access, worms, and viruses, all of which pose a threat to the stability of the operating system. A basic tenet of security is the concept of authentication, or “proving you are who you say you are”. Several forms of authentication exist, such as a user/password combination. Simply put, a user needs to enter his/her assigned password in order to access the system. This proves that a user is who they say they are. It is also possible to require use of a physical key (keycard or other access device) or a generated software key that is read by the system. Finally, biometrics may be implemented where a user’s fingerprint or retina are scanned into the system and checked against a database. Many threats to a system’s integrity exist to oppose these security measures. A Trojan Horse somehow gains access to a user’s credentials, which can then be sent to a malicious hacker (who can later use them to log in as a legitimate user). A Trap Door is, simply put, a built-in security vulnerability. Trap Doors can sometimes be intentionally created for monitoring purposes, but if a malicious user is made aware of the method of access, it becomes a problem. A Logic Bomb is much rarer, but just as alarming. A Logic Bomb is a security vulnerability that exists in an otherwise safe piece of software. Said vulnerability only shows itself when certain conditions are met. The extremely conditional nature of this threat can make it difficult to detect. A virus is a malicious piece of software that replicates itself in order to avoid deletion. It attempts to damage the system it’s installed on, all the while avoiding attempts at removal. A worm is like a virus, but it attempts to squander system resources in order to bring the affected system to a halt. Port scanning can be used to find vulnerable access points in an online system, and finally, DDoS attacks (Direct Denial of Service) are bruteforce attacks where a system is bombarded with large amounts of requests for access. This overloads the system and renders it inoperable until it can be restarted. In conclusion, security measures must be built into an Operating System, otherwise sensitive data can be manipulated by malicious users.

Report 9: Anil Singh, Remote Method Invocation in Enterprise Java Beans

A client-server model at enterprise level needs transaction management, security, networking, resource management, persistence, messaging, and deploy-time customization. Enterprise Java Beans uses stubs to have a client call business methods on an object that lives on the server. Stubs use Java’s distributed technology for remote method invocation. In enterprise java beans, the client calls a method on a stub, and that stub takes care of low-level communication (sockets, streams) with a remote object. It is also necessary to create a “skeleton” on the server side as well, otherwise networking code goes into a remote object, and that defeats the purpose of what we’re trying to accomplish. In order to handle method arguments and return values, stubs on the client/server side handle packing/unpacking of arguments. Enterprise java beans has some limitations. Not all datatypes can be arguments or return items. It is only possible to work with primitive datatypes, serializable objects, arrays of these objects, or a remote object. There are 3 kinds of beans in Enterprise Java Beans: Entity Beans: A thing in

persistent storage. Message-driven Beans: Listens to messaging service. Session Beans: Everything else, typically processes. Session Beans can be Stateful : Remembers

conversational state. Stateless : Forgets about client once method completes.

Report 10: Philip Lindner, Parallel Filesystem for Large-scale Datasets

A parallel filesystem is different from a traditional one in that it spreads data across multiple disks. Reading and writing is done in parallel across these disks. Generally, a parallel filesystem is used by a parallel computing system. Multiple nodes in the parallel system can access data

concurrently. Parallel filesystems are generally used in High Performance Computing, where their efficiency saves valuable time that is better used running intensive calculations or simulations. It’s important to remember that parallel file systems are different from distributed ones. Since both filesystems use multiple drives in one system, it can be easy to get the terms mixed up. To tell a parallel system from a distributed one, it’s important to remember that a

distributed filesystem stores files on single storage nodes, while a parallel filesystem distributes contents of files across multiple nodes. Because of this, parallel filesystems are better suited for concurrent access of a file. HPC, or High Performance Computing, is a resource-intensive type of computing that benefits the most from a parallel filesystem. Since writing serially to a drive is not sufficient for the bandwidth needs of high performance computing, an alternative was needed. Accessing a single disk creates an I/O bottleneck, but parallel drives alleviate the problem, achieving high I/O bandwidth. Parallel filesystems must utilize integrity, bandwidth, stability, and resiliency. They utilize compute nodes and I/O nodes. File striping is used to split data into fixed size blocks. Performance can be improved by minimizing thenumber of drives that a process must communicate with, and this is accomplished by stripe aligning I/O requests. Parallel filesystems have a file spread across multiple drives, so if one drive fails, the entire system is useless. GPFS and Lustre are two real-world examples of parallel filesystems. In conclusion, A parallel filesystem spreads data over numerous disks, and parallel filesystems improve the I/O bottleneck so commonly seen in high performance computing.

Report 11: Aashiq Boga, Detailed Overview of GIT Commands and their Functionality

GIT is a type of Version Control System that is the best available due to its speed and simple user interface. It derives its name from its creator, Linus Torvalds, poking fun at himself. It uses data snapshots, and is distributed. GIT utilizes branching, which creates independent local branches, and takes only seconds to implement. GIT has many commands. GIT commit stores the current contents of the index in a new commit along with a log message from the user describing the changes. When recording your own work, the contents of modified files in your working tree are temporarily stored to a staging area called the "index" with git add. A file can be reverted back, only in the index but not in the working tree, to that of the last commit with git reset HEAD -- <file>, which effectively reverts git add and prevents the changes to this file from participating in the next commit. After building the state to be committed incrementally with these commands, git commit (without any pathname parameter) is used to record what has been staged so far. GIT checkout updates files in the working tree to match the version in the index or the specified tree. If no paths are given, git checkout will also update HEAD to set the specified branch as the current branch. GIT branch is very simple, it does exactly what it is named to do – create a new branch. GIT merge is used by GIT pull to incorporate changes from another repository and can be used by hand to merge changes from one branch into another. GIT rebasing is similar to GIT merging in that it is another way to integrate changes from one branch into another. However, rebasing takes all the changes that were committed on one branch and replays them on another one.

Report 12: Sai Anuhya Sivanna Dommolapati, Mondrian Memory Protection

Mondrian memory protection is a fine grained memory protection scheme that allows multiple protection domains to flexibly share memory and export protected services. Although it can be layered on top of demand paged virtual memory, MMP is also well-suited to embedded systems with a single physical address space. Modern operating systems use a linear addressing scheme, in which each user process has a separate linear demand-paged virtual address space. Each address space has a single protection domain, shared by all threads that run within the process. Although this addressing scheme is now used everywhere in modern OS designs and hardware implementations, it’s not the best when used for protected sharing. To reduce the space and run-time overheads of providing finegrained protection, Mondrian memory protection uses a highly-compressed permissions table structure and two levels of hardware permissions caching. MMP meets three requirements of a good memory system: different: Different protection domains can have different permissions on the same memory region. small: Sharing granularity can be smaller than a page. Putting every network packet on its own page is wasteful of memory if packets are small. Worse, to give the header separate permissions from the payload would require copying them to separate pages in a page-based system (unless the payload starts at a page boundary). revoke: A protection domain owns regions of memory and is allowed to specify the permissions that other domains seefor that memory. This includes the ability to revoke permissions.

Report 13: Sri Raghunathan, Distributed System and the Communication Network

A distributed system is a collection of independent computers that appears to itsusers as a single coherent system. In a distributed system, differences between various computers and the way they interact are hidden from the users. Users and application can interact with the distributed system in a consistent and uniform way regardless of where and when the interaction takes place. As a whole, it is relatively easy to build on and scale up, if the need to enlarge it arises. To support heterogeneous computers and networks while offering a single system view, distributed systems are often organized by means of a layer of software that’s logically placed between the users and applications and layer underneath consisting of operating systems. The distributed system is sometimes known as “middleware”, since it resides between users and applications. Such middleware provides local resource management and communication mechanisms between users, applications and the operating system. It does not manage local nodes. Unlike homogenous systems, Distributed Systems are built on heterogeneous multicomputer systems. These systems widely vary with respect to processor type, memory sizes and I/O bandwidth. Heterogeneous Multi computer Systems has different complex nodes. The nodes may be few computers which are a part of the network, that may be a homogenous multicomputer. The operating system for a distributed system can be tightly coupled (tries to maintain a single, global view of resources) or loosely coupled (can be considered as computers running their own operating system still working together for providing services and resources available to users).