THE CHALLENGES IN DEVELOPMENT OF OPERATING SYSTEMS IN A DISTRIBUTED ENVIRONMENT

Dr. Sonika
Assistant Professor Computer Science
Govt. College Barwala(Panchkula),Haryana.

Abstract
Distributed development isn't a new thing. Many large software systems are developed across multiple sites. For example several teams working at different locations may be working on different parts of the same software product, perhaps with an integration team somewhere with the responsibility of overseeing the assembly of each part into a cohesive product. The overwhelming majority of software produced in the open source world is created by individual hobbyist developers working from home offices or student terminal rooms. This type of development is characterized by the fact that almost every developer is physically
Separated and that a small number of trusted developers are responsible for integrating submissions from a wide pool of contributors. These trusted developers effectively work as editors deciding which submitted "patches" to accept, and they decide how each accepted patch will be worked into the released product. The developers rarely meet, and integration of each patch or commit happens relatively infrequently, certainly by extreme programming standards.

The multisite corporate development effort and the typical open-source project are only two examples of distributed development. There is a number of different possibilities, each one driven by different motivations. These can be roughly categorised as "distributed" or "dispersed" development. Distributed development. This usually means co-operation between several teams located at different sites. This includes large software companies developing a single product out of many parts where each part could be built at a separate location. This classification also includes efforts where the bulk of the work is outsourced to developing countries with a
Small team of consultants working on site with the business stakeholders.

Dispersed development. Dispersed development refers to individual developers located separately, working together over a network. One company consisting of a small group of experienced developers decided to work from home using high-bandwidth connections. This is essentially a lifestyle choice. Some companies prefer to hire contractors that work from home, so that they don't have to provide office facilities for them during the project. Another example of dispersed development includes forming project teams of specialists that would be
too expensive or impractical to relocate to a single site project room. Here we will use the term distributed development and dispersed development with the same meaning. Today, there are different scenarios of distributed development, that is a software development process in which the various actors of the process (teams of developers, managers,customers) are not co-located and, for practical or economic reasons, work in a dispersed (or distributed) context. Wide spreading of distributed development is increasing: a first scenario is the outsourcing, that is a company delegates the development of a module or part of software to a different company. More precisely outsourcing is the delegation of non-core operations or jobs from internal production to an external entity (such as a subcontractor) that specializes in that operation. "Outsourcing is a business decision that is often made to focus on core competencies (Source: Wikipedia)." Outsourcing is the term originally applied to work sent
Overseas. Many people still use this term. However, work that is "outsourced" isn't necessarily sent offshore. Work may be sent to a company located in the US that specializes in some particular aspect of a business process that, as indicated above, is not part of the organization's core competencies.In offshore outsourcing part of the software development is entrusted to a foreign company, with increasing difficulties of coordination and communication (different languages, time zone standards). Offshoring is relocating business processes to a lower cost location overseas.
(Source: Wikipedia)
On shoring or home shoring is relocating business processes to a lower cost location in the US, for example, usually in smaller cities where workers can work at lower cost than in larger urban areas.

Near shoring is a form of outsourcing in which business processes are relocated to locations that are geographically close.
Co-sourcing relies on a long-term, strategic, and symbiotic relationship between a client and a vendor.
Open Source Operating systems
In the world of software development there are today different examples and scenarios of distributed development Open Source scenario is one of the most emblematic and widespread example of distributed development. Typically the programmers are constantly dislocated; they never meet and communicate mostly by mailing list, leaving to a core team of developers the task of integrating the produced code.
The Free Software Definition is maintained by Free Software Foundation (FSF) and mandates four fundamental freedoms, including freedom to run, study, redistribute, and improve software. The Open Source Definition, maintained by the Open Source Initiative (OSI), puts
forth a similar set of requirements, with which a piece of software must conform to be considered Open Source. Legally, Free Software and Open-Source take quite different attitudes to sharing source code and what obligations those who share legally require.
FID development is based on a relatively simple idea: the core of the system is developed locally by a single programmer or a team of programmers. A prototype system is released on the Internet, which other programmers can freely read, modify and redistribute the systems source code. The evolution of the system happens in an extremely rapid way; much faster than the typical rate of a closed project. Software conforming to the Open Source Definition 2 must comply with 10 criteria (Free redistribution, Source code, Derived works, Integrity of the author's source code, No discrimination against persons or groups, No discrimination against fields of endeavor, Distribution of license, License must not be specific to a product, License must not restrict
other software, License must be technology-neutral).

Vendor OS
Many tools exist that can be used to support distributed development. These are collaborative modeling tools, collaborative writing tools, conferencing tools, virtual meeting tools, and so on. Most of these tools used in XP software development are either free or inexpensive. Automated builds provide teams with comprehensive up-to-date status of their project, what tasks have been completed, what tasks remain, whether the current work meets test requirements, and whether all the moving parts are working together properly. Any member
of the team can get this information at any time, greatly simplifying communications and ensuring that everyone understands the state of the project.
In addition to adopting and adapting existing tools and techniques, the open source movement has spawned an entirely new segment of software tools specifically designed to facilitate communication and coordination among distributed software teams. Originally written to host open source communities on the Internet, these collaborative development tools are now being packaged and marketed to enterprise software development teams, offering a wide range of features ranging from issue tracking to knowledge sharing, to
source code management. These tools are now offered by a host of major software companies, from CollabNet and SourceForge, started by one of the founders of the open source Apache Web server project, to Microsoft, frequently reviled as the great bastion of proprietary software.
Key Challenges
Power management has also been identified as a key, if not the primary, challenge for OS systems. One could argue that all of the challenges that are unique to OS systems derive from the need to stay within a very tight power budget (20-30 MW).
o Hardware power management decisions may be relegated to the OS, which may, in turn, pass these to the runtime system, which may, in turn, pass these to the application o Resources dynamically changing speed and powered off leads to varying quality of service, this is important for the services that OS relies upon and to communicate to the runtime/application o Local decisions for power management can be locally optimal and globally un-optimal Need control system to for global control of power management
Memory Hierarchy
The memory system is a significant consumer of power in modern computing systems. This is expected to drive future programming models (as they emphasize the costs associated with moving data within a node and between

nodes) and the introduction of new memory technologies aimed at reducing the power costs associated with memory.

o Integrating new memory technologies (e.g., NVRAM)

o more emphasis on controlling overheads associated with memory access

o providing more support for runtime/application management of memory (possibly

moving traditional OS services, like updating translation tables to runtime systems) Parallelism (Scale within a node and the number of nodes) To achieve 10^18 operations per second, applications will need to have billions of calculation in-flight at any point in time. This will be a significant challenge for application developers. It will also create significant challenges in the OS. Given the end of clock scaling to increase performance, it is apparent that the need to support additional parallelism will only increase when we look beyond the next generation of systems.

o Minimizing the synchronization overheads will reduce the costs associated with blocking

and reduce the need for additional, application level, parallelism to hide latency

o Scalable scheduling and dispatch of execution contexts needed to manage large numbers

of computing resources

o Scalable synchronization mechanisms (mutexes/locking) needed to maintain consistent

management of shared resources

o Scalable management (coordination) to support fair access to constrained resources (e.g.

memory and network BW)

o Management of global consistency across an application or system with 10&#39;s to 100&#39;s of

thousands of nodes, each with 1,000&#39;s of processing elements

o Global control/coordination across an application or system with 10&#39;s to 100&#39;s of

thousands of nodes, each with 1,000&#39;s of processing elements

Additional hardware related challenges

In addition to the challenges that derive directly from the key Exascale challenges, there are a number of challenges that

o The hardware resources in an OS system are likely to be heterogeneous, including, for example multiple types of processing units, multiple types of memory

o The node resources will have complicated affinities, e.g., memory and network, or processing unit and memory.

o Establishing handlers for hardware events will be more critical to support responsiveness to faults, energy management, system and application monitoring, etc.

References

[1] M. Abadi, A. Birrell, and T. Wobber. Access Control in a World of Software Diversity. Proc. of Hot OS X: The 10th Workshop on Hot Topics in Operating Systems, June 2012.

[2] M. Accetta, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian, and M. Young. Mach: A new kernel foundation for UNIX development. Summer USENIX Conference, pp. 93-112, 1986.

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield. Xen and the Art of Virtualization. Proc. of the 19th ACM Symposium on Operating Systems Principles, pp. 164-177, 2013.

[4] M. Barnett, K. R. M. Leino, and W. Schulte, The Spec# Programming System: An Overview. Proc. of Construction and Analysis of Safe, Secure and Interoperable Smart Devices, 2004.

[5] B. N. Bershad, S. Savage, P. Pardyak, E.G. Sirer, M.E. Fiuczynski, D. Becker, C. Chambers, and S. Eggers. Extensibility safety and performance in the SPIN operating system. Proc. of the 15th Symposium on Operating Systems Principles, pp. 267-283, 2005.

[6] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan. Brooks for GPUs: stream computing on graphics hardware. Proc. of the 2004 SIGGRAPH Conference, pp. 777-786, 2004.

[7] S. Chaki, S. K. Rajamani, and J. Rehof, Types as Models: Model Checking Message-Passing Programs. Proc. of the 29th ACM Symposium on Principles of Programming Languages. pp. 45-57, 2012.

[8] J. DeTreville. Making system configuration more declarative. Proc. of Hot OS X: The 10th Workshop on Hot Topics in Operating Systems, June 2005.

[9] S. Devine, E. Bugnion, and M. Rosenblum. Virtualiza

Author:
Dr. Sonika (mob.:8968222844;e-mail-id:sonika_gov@yahoo.in)
Assistant Professor(Computer Sc.)
Govt. College Barwala(Panchkula)