Research supported tracing-based bad smells

CPU hog

A CPU hog refers to a process that spends too much time occupying the CPU. As a result, other processes are being starved, meaning that they do not get enough CPU time and take way too long to complete. CPU hogging is a problem because it will cause slowdowns in the entire system. CPU hogging can be prevented with an effective CPU scheduling algorithm; however, endless loops or processes in a chain of circular waiting can cause it to occur.

In *Detecting anomalies in microservices with execution trace comparison,* the authors test for this bad smell using their call tree approach. Because CPU hog greatly increases the latency of the system, the authors found that their ability to detect latency anomalies was very useful. It was tested for using Social Network in DeathStarBench. The authors created the smell by applying StressLinux, which added additional computationally heavy operations to the Social Network application.

https://www-sciencedirect-com.proxy.library.brocku.ca/science/article/pii/S0167739X20330247

Network congestion

Network congestion refers to issues that occur when a node in a network is carrying more data than it was meant to handle. It can reduce the quality of the services being provided or cause a denial of the services. Network congestion often occurs due to malicious intent, where attackers overload the network and cause it to fail.

In *Detecting anomalies in microservices with execution trace comparison,* the authors test for this bad smell using their call tree approach. Once again, because network congestion causes a large increase in latency, the authors were able to use their latency anomaly detection to accurately test for this bad smell. It was tested for using Social Network in DeathStarBench. The authors used iPerf to send large amounts of packages to the network, occupying 75 to 90% of the bandwidth.

https://www-sciencedirect-com.proxy.library.brocku.ca/science/article/pii/S0167739X20330247

Memory leak

A memory leak refers to the phenomenon where memory that is no longer needed is not released to make room for more data. Eventually, the program runs out of memory. Because they are a problem that builds up over time, memory leaks are a common cause for software aging. Memory leaks are a problem because it can be difficult to detect their source and they are a severe waste of resources.

In *Detecting anomalies in microservices with execution trace comparison,* the authors test for this bad smell using their call tree approach. This approach was not able to detect a memory leak right away. Once the problem became serious, however, it was detectable by their approach. It was tested for using Social Network in DeathStarBench. The authors implemented this bad smell by simply allocating objects to memory and never releasing them.

https://www-sciencedirect-com.proxy.library.brocku.ca/science/article/pii/S0167739X20330247

Service hangup

Server hangup refers to the phenomenon where the server the computer is communicating with has stopped responding. It could be caused by a number of different problems, like a disconnected internet service. This issue can result in a service failure. There is not much else to describe about this one.

In *Detecting anomalies in microservices with execution trace comparison,* the authors test for this bad smell using their call tree approach. The approach was able to easily detect this bad smell because it causes both tree structure and latency anomalies. It was tested for using Social Network in DeathStarBench. The authors implemented this bad smell by deleting the deployment controller of Kubernetes, the system they used to manage the applications in their test.

https://www-sciencedirect-com.proxy.library.brocku.ca/science/article/pii/S0167739X20330247

Deadlock

A deadlock refers to a scenario where every process in a group is waiting for another process in the group to release a resource. In this situation, no process can continue and will just wait for its resources forever, unless the problem is fixed. Obviously, this is a major problem for the system. There are many different situations that can lead to a deadlock.

In *Detecting anomalies in microservices with execution trace comparison,* the authors test for this bad smell using their call tree approach. The authors were able to use the latency and structure anomalies to detect this bad smell. It was tested for using four microservices in Bench4Q: Buy confirm, Buy Request, Search request, and Product detail. The bad smell was implemented creating badly designed database tables. For example, the authors put an exclusive lock on a table and adding the statement SELECT ... FOR UPDATE.

https://www-sciencedirect-com.proxy.library.brocku.ca/science/article/pii/S0167739X20330247

High response time (microservices)

High response time in microservices refers to the phenomenon where the microservice takes longer than it should to act on a user's request. Once again, there are many potential causes for the high response time. In *Detecting anomalies in microservices with execution trace comparison*, the authors specifically explain that this error can come about because of improper configuration of the microservice.

In *Detecting anomalies in microservices with execution trace comparison,* the authors test for this bad smell using their call tree approach. They were not able to detect this bad smell easily using structure anomaly detection because there were very few changes to the trace call tree even though the response time was higher. However, the latency anomaly detection was able to detect this bad smell. It was tested for using four microservices in Bench4Q: Buy confirm, Buy Request, Search request, and Product detail. The authors implemented this bad smell by setting the "maxActive" field in the configuration file equal to "10", when it should have been much higher.

https://www-sciencedirect-com.proxy.library.brocku.ca/science/article/pii/S0167739X20330247

Bad smell categories

Synchronization bad smells

These bad smells have to do with parallelism, and issues that can occur from the use of threading. Examples include deadlock, livelock, lock contention, and race conditions.

https://www.imperial.ac.uk/media/imperial-college/faculty-of-engineering/computing/public/1819-ug-projects/Davies-LyonsA-IO-bottleneck-detection-in-the-Linux-kernel.pdf

I/O bad smells

These bad smells are related to the processes of reading data from external sources and writing data to external sources. Examples include inefficient disk accesses and fragmentation.

https://www.imperial.ac.uk/media/imperial-college/faculty-of-engineering/computing/public/1819-ug-projects/Davies-LyonsA-IO-bottleneck-detection-in-the-Linux-kernel.pdf

Unsupported bad smells

Inefficient disk access

During runtime, the system frequently needs to access the hard disk in order to obtain information that is not currently being stored on RAM. Additionally, part of the data stored in RAM will need to be written to the disk when this happens, because more room will be needed in the RAM. Moving the read/write head to the correct location is the most time-consuming part of this operation. Because RAM access does not feature any moving parts, these types of accesses are much more efficient than disk accesses. Obviously, the CPU should try to minimize disk accesses in order to save time. When the system performs inefficient disk accesses, it is performing more disk accesses than are necessary for an efficient system.

Software aging

Software aging refers to the phenomenon that sometimes occurs where the longer a piece of software runs, the more problems it encounters. The software will initially run with no issues but will run into issues as time progresses. Software aging is a problem that is difficult to diagnose and explain, due to the fact that it can have many different causes, and these causes take some time to reveal themselves.

Buffer overflow (or Stack buffer overflow)

A buffer overflow can occur when the system attempts to write more data to a buffer than the buffer can hold. The extra data gets spilled over and overwrites adjacent memory addresses. This can cause memory access errors and crashes. Additionally, attackers can use this method to inject code into a program, which is a huge security issue. Buffer overflow is also a form of stack corruption.

Buffer overread

A buffer overread is very similar to a buffer overflow, except it occurs during the read operation. The receiver attempts to read more data from the buffer than the buffer can hold. If the buffer boundary checking contains loopholes, then the data in the memory addresses adjacent to the buffer will also be read. This is an issue because it can be used to leak confidential information.

Dangling pointer

A dangling pointer is a pointer which no longer points to the correct data in memory. The data it pointed to was deallocated while the pointer was allowed to remain existing. This type of error would be difficult to exploit. However, it could be used in a manner similar to a buffer overflow. It also reduces the software quality because it can make the program unstable.

Stack corruption

This type of security issue occurs when some locations of memory on the stack get overwritten with new data. There are multiple causes for this phenomenon, including a buffer overflow. It can cause software to do unpredictable or unexpected things, meaning that it is a security issue when it occurs in programs.

Fragmentation (internal, external)

Fragmentation occurs when space in memory is wasted because data in memory does not occupy adjacent spaces. This is both an inefficient use of storage space as well as a cause for higher disk latency, because it can force files to be stored in an incontiguous location in memory, causing more read operations to be necessary. Internal fragmentation occurs when extra memory is allocated where it is not needed. This leads to a space in memory that is not being used. External fragmentation occurs when there is enough memory for the required task, but it is not contiguous. As mentioned above, this can lead to performance issues like inefficient disk accesses.

Unreachable memory

Unreachable memory is very closely connected with a memory leak and happens when all references to a particular point in memory are deleted. Most languages will reclaim memory that has become unreachable, referred to as garbage collection. However, unreachable memory that is not reclaimed will lead to a memory leak.

Starvation

When the CPU is scheduled poorly, certain processes never get CPU time. Therefore, they never get finished. A process that does not get enough CPU time to finish is said to be starved. Starvation is a problem because processes that are starved hold on to valuable resources and may be required to be completed before other processes can be started. Starvation can be caused by an inappropriate priority scheduling technique, where lower value processes are always blocked by higher value processes.

ABA problem

The ABA problem refers to a synchronization problem where communication between two synchronous processes is confused. Let’s say process one is in state A when process two checks it. Next, process one switches to state B and back to state A before process two checks it again. Process one has changed states, but process two does not realize it. This is the ABA problem. If it goes unchecked, it can create race conditions.

Livelock

A livelock is similar to a deadlock in that it is a situation where the system cannot make any progress. In a livelock, however, all processes are not waiting for another process to release a resource. Instead, the system is stuck in an infinite loop where they repeat the same actions over and over again in response to changes made by other processes stuck in the livelock.

Producer-consumer problem

The producer-consumer problem really is concerned with the idea of the critical section of code. In the case of the producer-consumer problem, the critical section is the buffer that the producer is writing to and the consumer is reading from. The two should not both try to access the buffer at the same time because it will lead to unpredictable results. This is a common example of a race condition.

Race condition

The race condition can occur when the outcome of a task is determined by the order in which multiple processes complete their portion of the task. If this is the case, the order that the processes complete their operations must be set beforehand, otherwise a race condition will occur.

Lock contention

Lock contention happens when two processes attempt to access the same lock. Only one is able to access the lock at one time. Because of this, in a concurrent system where locks are implemented poorly, all possible benefits of parallelism will be removed and the system will have poor performance.