ANTI PATTERNS FROM Dr. Vishwanath Rao

Falling Dominoes

PROBLEM

Occurs when one failure causes performance failures in other components.

SOLUTION

Make sure that broken pieces are isolated until they are repaired.

Empty Semi Trucks

PROBLEM

Occurs when an excessive number of requests is required to perform a task. It may be due to inefficient use of avail- able bandwidth, an inefficient interface, or both.

SOLUTION

The Batching performance pattern combines items into messages to make better use of available band- width. The Coupling performance pat- tern, Session Facade design pattern, and Aggregate Entity design pattern provide more efficient interfaces.

Roundtripping [Tate 2002]

PROBLEM

Special case of Empty Semi Trucks. Occurs when many fields in a user interface must be retrieved from a remote system.

SOLUTION

Buffer all the calls together and make them in one trip. The Facade design pattern and the distributed command bean accomplish this buffering.

Tower of Babel

PROBLEM

Occurs when processes excessively convert, parse, and translate internal data into a common exchange format such as XML.

SOLUTION

The Fast Path performance pattern identifies paths that should be stream- lined. Minimize the conversion, pars- ing, and translation on those paths by using the Coupling performance pat- tern to match the data format to the usage patterns.

Unbalanced Processing [Smith and Williams 2002]

PROBLEM

Occurs when processing cannot make use of available processors, the slow- est filter in a “pipe and filter” architec- ture causes the system to have unacceptable throughput, or when extensive processing in general impedes overall response time.

SOLUTION
---

1) Restructure software or change scheduling algorithms to enable con- current execution.
2) Break large filters into more stages and combine very small ones to reduce overhead.
3) Move extensive processing so that it doesn't impede high traffic or more important work.

Unnecessary Processing [Smith and Williams 2002b]

PROBLEM

Occurs when processing is not needed or not needed at that time.

SOLUTION

Delete the extra processing steps, re- order steps to detect unnecessary steps earlier, or restructure to delegate those steps to a background task.

The Ramp [Smith and Williams 2002b]

PROBLEM

Occurs when processing time increases as the system is used.

SOLUTION

Select algorithms or data structures based on maximum size or use algo- rithms that adapt to the size.

Sisyphus Database Retrieval Perfor- mance Antipattern [Dugan, et al. 2002]

PROBLEM

Special case of The Ramp. Occurs when performing repeated queries that need only a subset of the results.

SOLUTION

Use advanced search techniques that only return the needed subset.

More is Less [Rogers and Boyer]

PROBLEM

Occurs when a system spends more time “thrashing” than accomplishing real work because there are too many processes relative to available resources.

SOLUTION

Quantify the thresholds where thrash- ing occurs (using models or measure- ments) and determine if the architecture can meet its performance goals while staying below the thresh- holds.

“god” Class [Smith and Williams 2002]

PROBLEM

Occurs when a single class either
1) performs all of the work of an applica- tion or
2) holds all of the application’s data. Either manifestation results in excessive message traffic that can degrade performance.

SOLUTION

Refactor the design to distribute intelli- gence uniformly over the application’s top-level classes, and to keep related data and behavior together.

Excessive Dynamic Allocation [Smith and Williams 2002]

PROBLEM

Occurs when an application unneces- sarily creates and destroys large num- bers of objects during its execution. The overhead required to create and destroy these objects has a negative impact on performance.

SOLUTION
---

1) “Recycle” objects (via an object “pool”) rather than creating new ones each time they are needed.
2) Use the Flyweight pattern to eliminate the need to create new objects.

Circuitous Treasure Hunt [Smith and Williams 2002]

PROBLEM

Occurs when an object must look in several places to find the information that it needs. If a large amount of pro- cessing is required for each “look,” per- formance will suffer.

SOLUTION

Refactor the design to provide alterna- tive access paths that do not require a Circuitous Treasure Hunt (or to reduce the cost of each “look”).

One-Lane Bridge [Smith and Williams 2002]

PROBLEM

Occurs at a point in execution where only one, or a few, processes may con- tinue to execute concurrently (e.g., when accessing a database). Other processes are delayed while they wait for their turn.

SOLUTION

To alleviate the congestion, use the Shared Resources Principle to mini- mize conflicts.

Traffic Jam [Smith and Williams 2002]

PROBLEM

Occurs when one problem causes a backlog of jobs that produces wide variability in response time which per- sists long after the problem has disap- peared.

SOLUTION

Begin by eliminating the original cause of the backlog. If this is not possible, provide sufficient processing power to handle the worst-case load.