November 09, 2018

Anomaly detection within software build processes

Michael Butler

Tresa Johnson

John Micco

Anomaly detection within software build processes

ABSTRACT

The building of software packages is a compute-intensive process, especially for relatively large software. Changes that can detrimentally affect the build process are often not detected until resource consumption impact reaches critical levels. These changes become increasingly difficult to detect and isolate as time passes from the date of the change, resulting in capacity crunches and an overall slowdown of the build process. This disclosure provides techniques to automatically detect resource consumption anomalies in the build process either in real time or soon after such anomalies arise. Anomalous events, e.g., relatively large changes in resource consumption not attributable to changes in the number of users or projects, are detected based on time series data for software projects that are being built.

KEYWORDS

software build; repository; package; anomaly detection; resource consumption

BACKGROUND

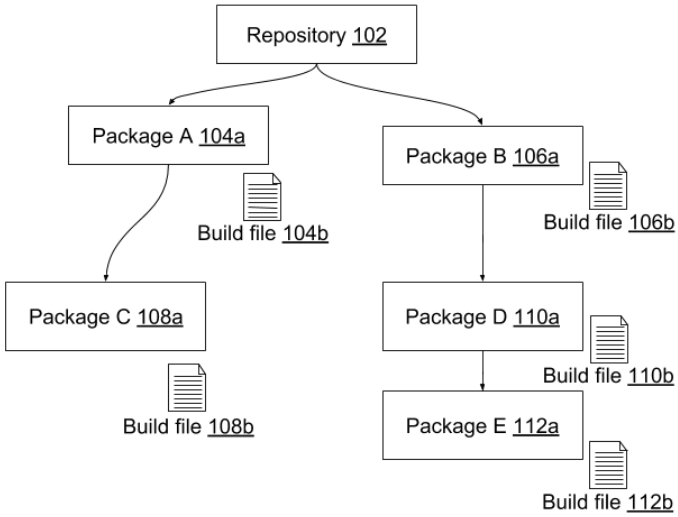

Fig. 1: Code organization in a software repository

Fig. 1 illustrates typical organization of a software repository. The repository (102) is organized into a tree of packages (104a-112a), each with an associated build file (104b-112b). Some packages may depend on or be derived from other packages, e.g., package E depends on package D, which in turn depends on package B, etc.

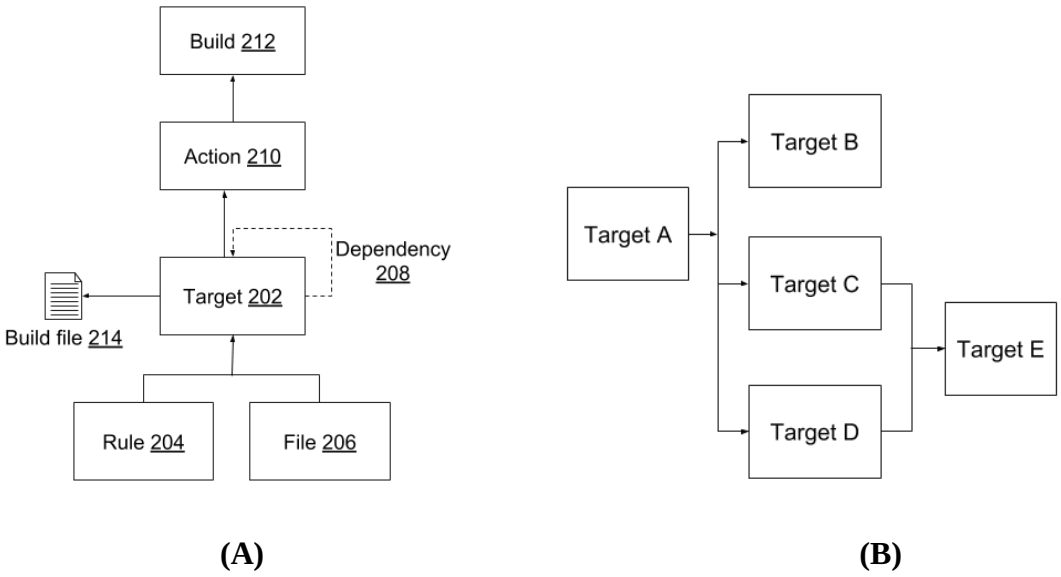

Fig. 2: A build file comprises a set of targets

Fig. 2A illustrates the composition of a build file. A build file (214) typically comprises one or more targets (202), a target being associated with rules (204) and files (206). A target itself may depend on other targets, as illustrated in Fig. 2B, where target E is a dependency for targets C and D, which are in turn dependencies (along with target B) for target A. Building of a target includes building of the dependency graph of the target. A target's dependency may lie in its own or in other packages. A build of a single target is considered as a separate action (210), wherein an action includes linking, compiling, executing, etc. Actions taken to build a final target belong to a single build (212), and reference it via a build-ID.

The build process can take place in the cloud, especially for relatively complex software. Information about the execution of actions is stored in a table in the cloud. Actions consume the

resources, e.g., memory, compute power, etc. of the cloud to execute. Resource consumption is measured in terms of service units.

Changes that can detrimentally affect the build process are often not detected until resource consumption reaches critical levels and begins to negatively impact users of the build system. Such changes become increasingly difficult to detect and isolate as time passes from the date of the change, resulting in capacity crunches and an overall slowdown of the build process. Live or early detection of resource consumption anomalies within a large multivariate dataset such as a build process is traditionally problematic, as initial changes to execution time due to anomalies are often small. The impact does become noticeable and even powerful over time, e.g., over months, at which point they are difficult to isolate.

DESCRIPTION

This disclosure provides techniques to automatically detect resource consumption anomalies in the build process either in real time or soon such anomalies they arise. Anomalous events, e.g., relatively large changes in resource consumption not attributable to changes in the number of users or projects, are detected based on time series data for software projects that are being built.

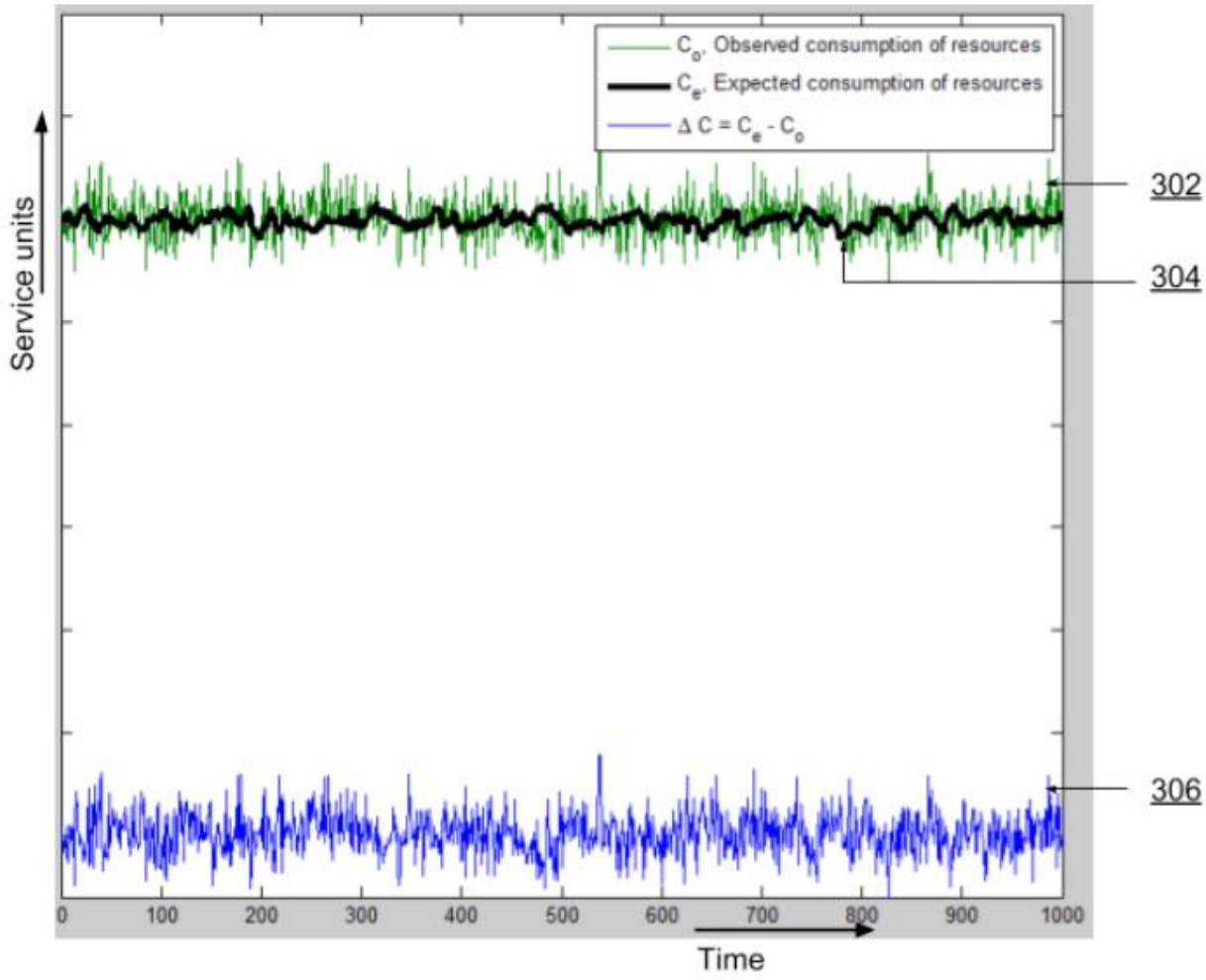

Fig. 3: Resource consumption versus time

Fig. 3 illustrates an example of resource consumption time series data. The green curve (302), labeled C_o, is the observed consumption of resources versus time. The observed consumption of resources is similar to a Gaussian process. The thick black curve (304), labeled C_e, is the expected consumption of resources. The expected consumption of resources C_e is predicted, e.g., using a recursive neural network. The blue curve (306), labeled ΔC, is the difference $C_o - C_e$ between the observed and expected consumptions of resources. Under normal conditions of service load, ΔC is close to zero.

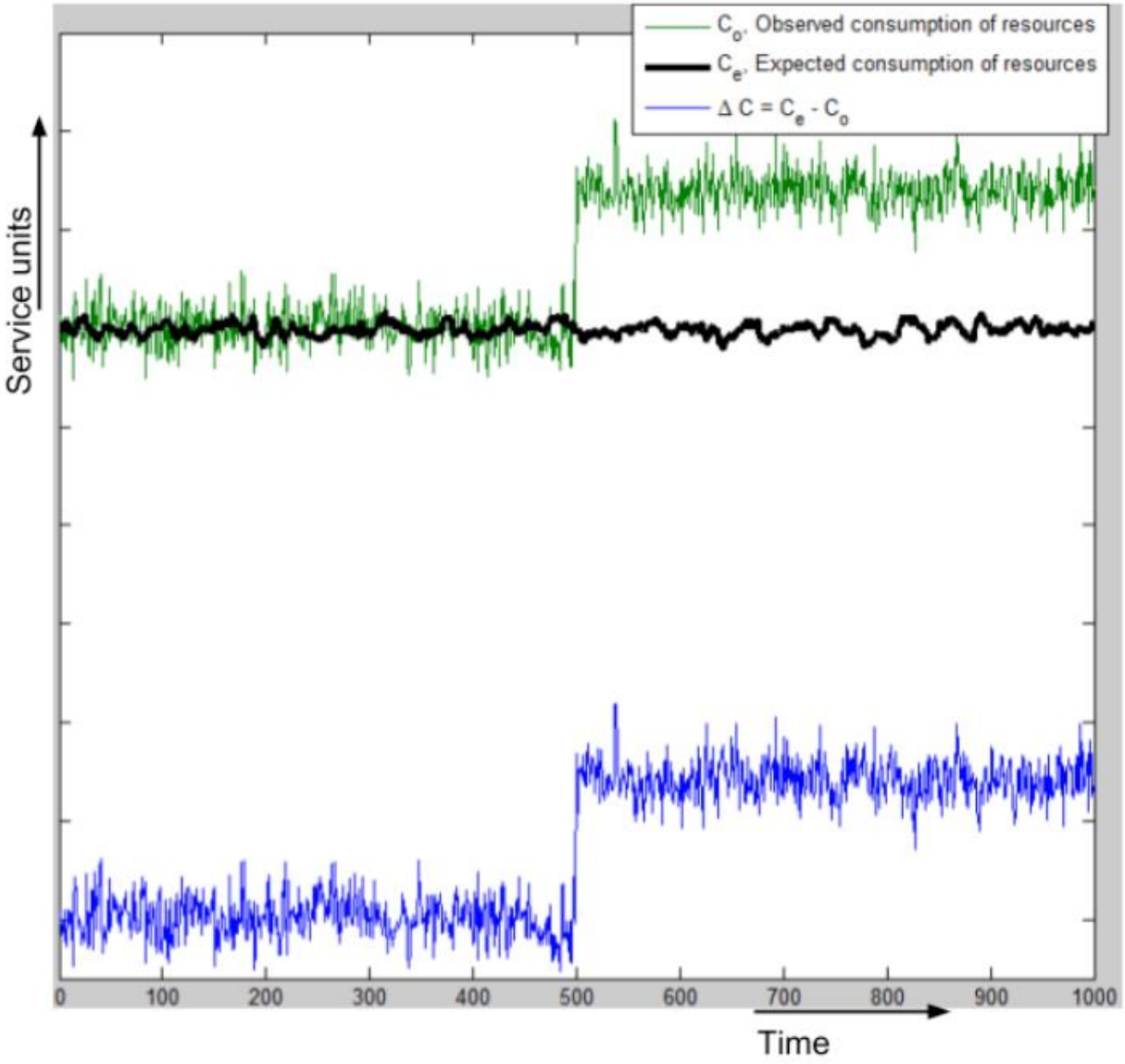

Fig. 4: A jump in observed consumption of resources

Fig. 4 illustrates a jump in the observed consumption of resources. The expected consumption of resources (black curve) remains flat, but due to a jump in observed consumption (green curve), the difference between expected and observed consumptions (blue curve) also experiences a jump. For readability, the jump in Fig. 4 has been exaggerated; in actual practice, the jump level need not be as pronounced, although it may be persistent with time.

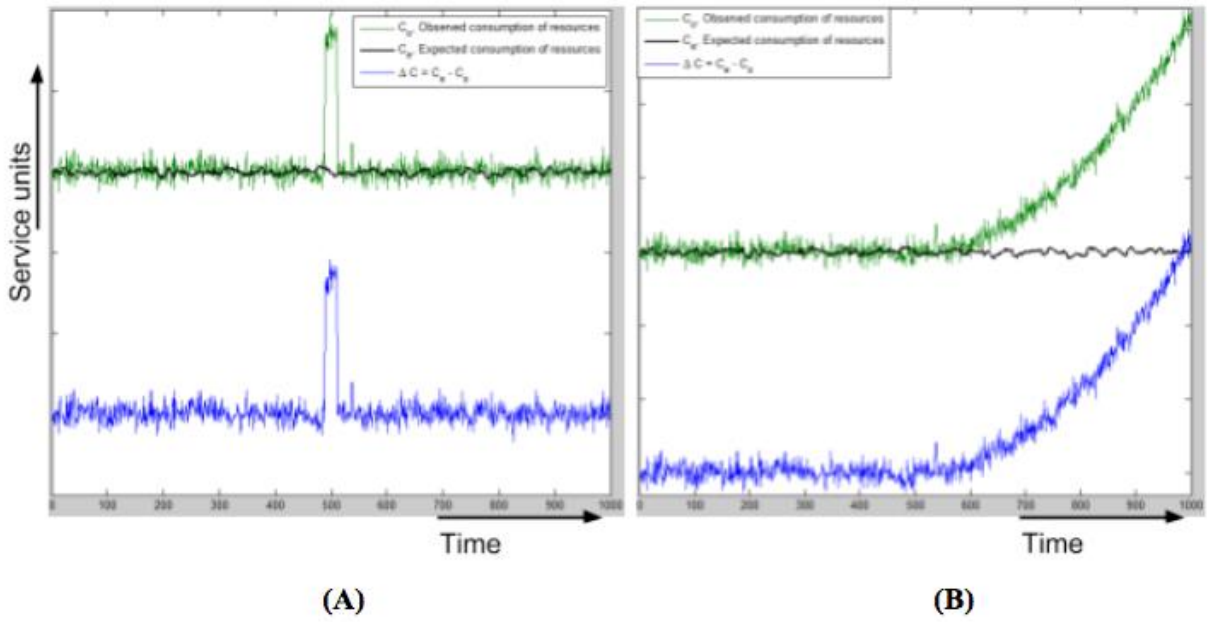

Fig. 5: Other forms of increases in computational resources: (A) spike (B) linear/super-linear increase

Fig. 5 illustrates other forms of increases in computational resources with time, e.g., a spike (Fig. 5A), a linear or super-linear growth (Fig. 5B), etc.

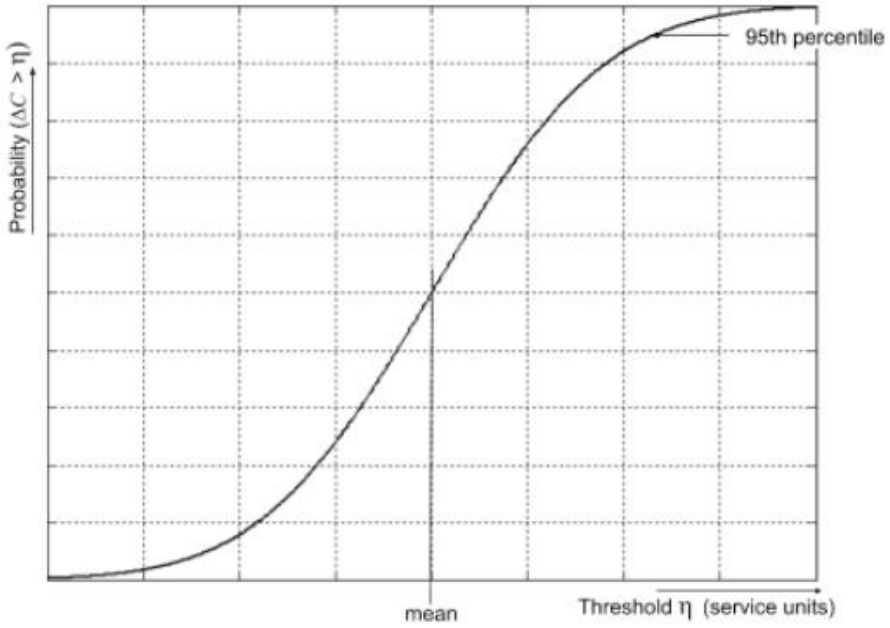

Fig. 6: Cumulative density function for Δ*C*

To determine anomalous events, a cumulative density function (CDF) of ΔC is computed over a time window prior to the current time. Fig. 6 illustrates an example CDF for ΔC. A threshold η is set for alerts based on the CDF. For example, an anomaly may be considered as an event where ΔC exceeds its 95 percentile.

When an anomalous event is detected, the software project(s) that are responsible for the event are determined by examining build processes that were active at the time of the anomaly. Alerts and/or bug reports are sent to the relevant developer team. The alert provides information relating to code in the software project that created the anomaly, and also provides the change-list where the event occurred.

Alternately, the first and second derivatives of time series data can be used to detect the anomalies. Using the first derivative enables sensitive detection of an anomaly. Using the second derivative enables the detection of presently small anomalies that over time have a significant impact.

Anomaly detection, per the techniques of this disclosure, can be used to automatically monitor aspects of the build system that are influenced by user behavior. Being able to detect these changes in time series data enables pre-emptive or early diagnosis of potential problems before they rise to a level of criticality. Automated detection and alerting of anomalous events in the build process enables the build service to scale economically, e.g., by enabling accurate provisioning and capacity projections.

CONCLUSION

This disclosure provides techniques to automatically detect resource consumption anomalies in the build process either in real time or soon after such anomalies arise. Anomalous events, e.g., relatively large changes in resource consumption not attributable to changes in the

number of users or projects, are detected based on time series data for software projects that are being built.