Post-Crash Analysis of the InnoTime Application

by Nathan Chung

Introduction

During the release of a mandatory update for the InnoTime smartphone client application, simultaneous database queries triggered by thousands of client instances crashed the database and caused the clients to lock up. This document describes the technical background, the failure incident, and the initial steps the teams took to remediate the problem. It concludes with future steps to prevent similar problems.

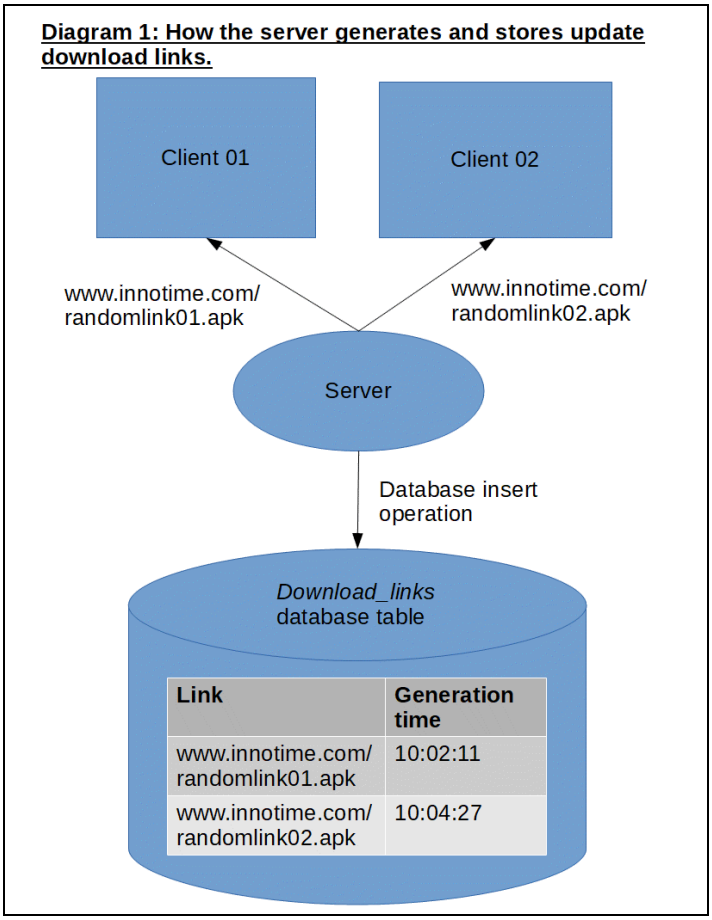
Technical Background

The InnoTime smartphone client is a business application that retrieves data from a central server. Whenever the client has a new update or is missing any required components, it receives the download links from the server. The steps on how the InnoTime server generates the links are as follows:

1. When the InnoTime server receives a download request from a client, it dynamically and randomly generates the download link, making each link unique to each client instance.
2. The InnoTime server sends the new link to the client.
3. The InnoTime server inserts a new record containing the download link and its generation time into a database table named *Download_links.*

The following diagram describes the above steps in a visual form:

Diagram 1: How the server generates and stores update download links.



A row in the *Settings* database table defines the link expiration period in minutes. The *Settings table* is composed of rows with two columns named *Name* and *Value*. Each row represents the name and the value of a setting. The row for the link expiration period setting has the name *Download_link_expiration*, and its value at the time of writing this document is set to 30 minutes.

When the client calls the server using the link, the server checks for the validity of the link with the following steps:

1. The InnoTime server queries the *Download_links* table for a row that contains the same link as the one sent by the InnoTime client.
2. If the query does not produce a match, the client's link is invalid. The InnoTime server refuses to transfer the update file.
3. If the query produces a match, the server looks at the link's generation time in the matching row.
4. If the link's generation time is older than the value for *Download_link_expiration*, the InnoTime server treats it as an expired link and refuses to transfer the update file.
5. If the link's generation time is within the value for *Download_link_expiration*, the InnoTime server will transfer the update file to the client.

Note that if the client has an invalid or expired link, the user has to get a new link by pressing the Download buttons on the client again. If the client has a pending mandatory update, the user

would not be able to proceed past the update download screen until completing the mandatory update.

How the Failure Occurred

On the morning of the crash, the development team released a new client update. Thousands of clients attempted to download the update simultaneously, leading the server to make thousands of queries to the *Downloads_links* table. Each query did a full scan of the *Download_links table,* which contained hundreds of thousands of rows representing past downloads. Unable to handle the load, the database crashed and prevented most of the client instances from successfully downloading the mandatory update. Users were locked out of the clients due to the mandatory update requirement. Even if the clients were able to successfully download and install the update, they still would not have been able to connect to the InnoTime server due to the database crash.

The database crash had two main factors: an overly broad query and the database team never having cleaned the *Download_links* table since its creation. The full scan query for *Download_links* was unnecessary since only links that were generated in the past 30 minutes as defined by *Download_link_expiration* in the *Settings* table were valid.

A more significant problem was that the database team never cleaned the *Download_links* table since its creation. When the development team first released InnoTime, the table was small enough that simultaneous queries from many clients did not overload the database. With no purging process in place, the table continuously grew.

Initial Remediate Step

The database team purged older rows with expired links from the *Download_links* table to reduce the number of rows. On the server side, the development team deployed a quick and temporary fix with potential problems as explained in the Future Steps section. In the *Settings* table, the development team created a new row with the name *Query_range* and a value of 60 minutes. The development team then refined the database query for *Download_links* to only look at rows created in the past *Query_range* minutes. With the fixes in place, the teams were able to bring the application back online, and users were able to successfully update their clients.

Future Steps

To prevent similar problems from occurring in the future, the database team and the development team would need to perform a thoughtful analysis of how the client, server, and database interact with each other. More specifically for the *Download_links* table, which was

one of the main factors for the incident, the database team can implement further improvements such as:

1. Set up an automatic purging schedule for download links that have expired.
2. Partition the table based on days, so that the server only queries for the newest rows that have not expired.

The development team created a potential technical debt by introducing a new setting named *Query_range* into the *Settings* database table. The development team's rationale for introducing another setting was to "create a buffer and separate the business logic from the backend code." By creating two separate variables, the team intended to separate the code that queries the *Download_links* table from the code that checks for the link expiration.

While the idea sounds good on the surface, it is flawed because the value of *Query_range* depends on the value of *Download_link_expiration* to avoid producing too little or too many matches. If the development team decides to change the value of *Download_link_expiration* for whatever reason, it would need to remember to manually update *Query_range* accordingly. The team would likely benefit from having a discussion on whether introducing the new *Query_range* setting was preferable to using the existing *Download_link_expiration* setting.