УДК 004.4

***Zhyliaiev Yevhenii, a 1st year Master's Student,***
***Vakaliuk Tetiana, Ds.Sc. in Education,***
***Professor of the Software Engineering Department***
*Zhytomyr Polytechnic State University, Zhytomyr. Ukraine*

# ANALYSIS AND OPTIMIZATION OF DISTRIBUTED TRACING CONCEPTS

From the perspective of observing and managing modern distributed systems, distributed tracing is progressively viewed as more and more important part of it. Throughout development of the concept, it has undergone multiple different milestone, which shaped different aspects of it; most essential milestones can be associated with particular tracing tools and research related to them. Namely, it is Google's Dapper and related published paper, Zipkin and OpenTracing projects, which were based off Dapper, and its ideas and the most recent tracing platform called LightStep, which is currently being offered «as a service». From a practical point of view, exa-mining such tools and concepts they were dealing with, could help to define the very concepts of distributed tracing and which part of it are applicable to each respective practical case.

According to the paper published by Google on its tool, Dapper, request tracing is based on a very simple idea: there are special points in the system which constitute particular interest due to representing sort of «crossroads» of the computational flow, like a start of a new transaction. The trick is all about defining those points accurately and thoroughly to get the picture of the whole system. A trace, the definition of which was more elaborated in the OpenTracing API project, is the information that gives you the idea about the path of the transaction took in order to make its way in the system. In its turn, trace also can be broke down into smaller units, in both Dapper and OpenTracing project those are called «spans» – adjacent segments of work that are distinguished in regards to both naming and execution time range. As a premise for the next set of challenges which were standing in the way of distributed tracing development, it's important to emphasize that trace data should not be wrapped together with the request itself, since that would influence the sizing of the request and in case of growing of the system size over time, it could cause uncontrollable ballooning of the request size itself, since the trace data size is in direct correlation with the size of the overall system it's used to observe. Thus, as part of the best practices, it was defined by Google, that trace data should be stored locally and requested into the system only in case of need. Which in its turn created a requirement of having a single points of storage of all the trace-related data.

All that leads to the first major challenge when it comes to implemen-ting of a distributed tracing system, which is the very architecture of system against which such a system is going to be used. As an example, when applying a micro service architecture, a common way of implementing it is not using a single framework or even a single programming language, which is also known as «heterogeneous structure», which obviously mean that such a system lacks some shared part where it could store and operate all the trace-related data. The approach that was used by both Dapper and Zipkin was to use a homogeneous RPC (remote procedural call) network, which led to the origin of a tool called «service mesh», a dedicated infrastructure layer built into an app that provides a way to control how different parts of an application share data with one another. Currently, service meshes can be used to «insert» a distributed tracing tool into your system without changing any code withing it.

Yet another major challenge is the amount of data that tracing of a big distribute system requires, especially when you have to account for the system's growth over time. Again, if we are to resort to the Google's experience with Dapper, best practice in this regard would be considered «sampling» – processing only one trace out of a certain number, let us say, one thousand. Of course, it is important in this case to balance out the ratio of such sampling in a way that the gain of performance would not defeat the purpose of distributed tracing and it would still provide a meaningful picture of the system and its operational state.

As a conclusion, it is important to state that such a concept as distribu-ted tracing should utilize other important sub-concepts, like logging and monitoring, together with the related best practices. In addition, nowadays the tools and any new challenges concerning distributed tracing gain more traction and allow more people to be involved in the process. That's why, an important caveat would be that while Dapper's research is full of helpful insights, not everyone has to solve problems at the Google's scale and it's important to be able to determine your particular use case and be able to choose proper tool or maybe even tailor the existing systems to your needs and that's exactly why you need to know the underlying concepts and fundamentals behind the existing technology of distributed tracing.