## 1 Structure of the 1st Paper

Since we want to make the ongoing thread of work into a few publishable concrete projects, and eventually my thesis, I want to propose (following my conversations with Suresh) the following structure for the contributions of my first paper:

### 1.1 Formal introduction of a new notion of application correctness

The theoretical novelties of this paper should be minimal and are going to be mainly based on the analysis framework of Kartik's paper. In this paper, we should not present any of our ideas regarding transaction chopping (fine-grained isolation guarantee annotations) and our application repair approach.

In this paper, we should however, elaborate the notion of Eventual Serializability (ES) and verify our benchmarks against ES. This requires minimal modification to the current analysis framework and will result in a broader set of benchmarks that we can utilize.

### 1.2 Realization of the analysis framework

The major contribution of this paper should be the realization of Kartik's analysis method in an OCaml tool which allows users to write their applications in a DSL and automatically creates the Z3 input.

### 1.3 Benchmarking

We should apply the tool on a set of benchmarks (I'd say about 10 applications) and present the automatic analysis results. For example, we can automatically conclude that transaction T from application B is safe under guarantee G.

The guarantees that we should verify the transactions against, should include the known weak SQL isolation levels, and also a few additional guarantees from NoSQL databases (e.g. CC or distributed PSI[1]).

[1] I do NOT think we should consider database specific guarantees (such as Cassandra's light-weight transactions) and only present a handful of famous concurrency control mechanisms

## 1.4 Evaluation

The second main contribution of the paper should be empirical results showing the performance gain from assigning weak concurrency control mechanisms to benchmark applications. We can first compare the optimized versions to naive versions (where all txns are SER) on a reference centralized SQL database. Additionally, we can manually (for now) port the given applications into equivalent NoSQL versions (maybe for 2 databases, e.g. both Cassandra and Riak) and show the performance gain from geo-distribution, specially when concurrency control mechanisms are weak.