Enhancing Clinical Trial Simulations with Advanced Computational Techniques and Scalable Cloud Solutions

Guillaume Bouchard[1*], Eliott Tixier[1], **Jean-Baptiste Gourlet[1]** , **Frederic Cogny[1]**
* Corresponding author, [1] Nova In Silico, Lyon, France

Leveraging cloud computing and Just In Time compilation, Jinkō can run **large *in silico* clinical trial** in a matter of minutes

Cloud computing architecture

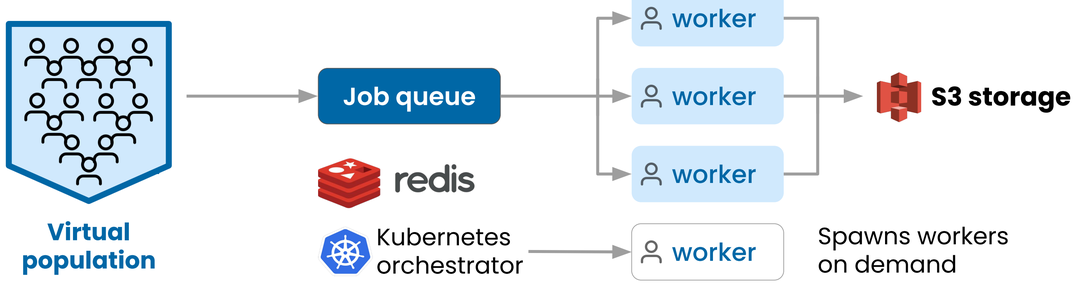

Figure 1: Jinkō's architecture for on demand scalability and parallelism

Just-in-time compilation

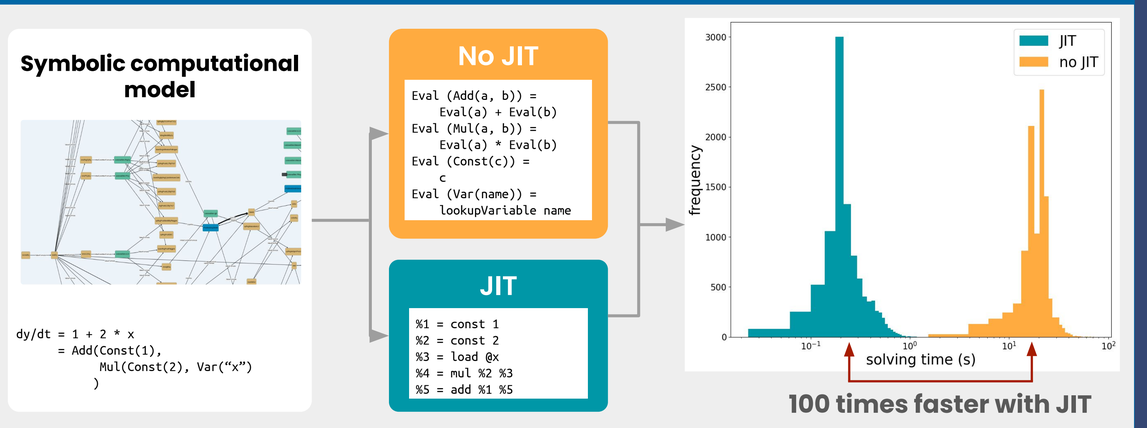

Figure 2: Just In Time compilation for faster solving

Computing time (wall clock) to simulate 30,000 patients

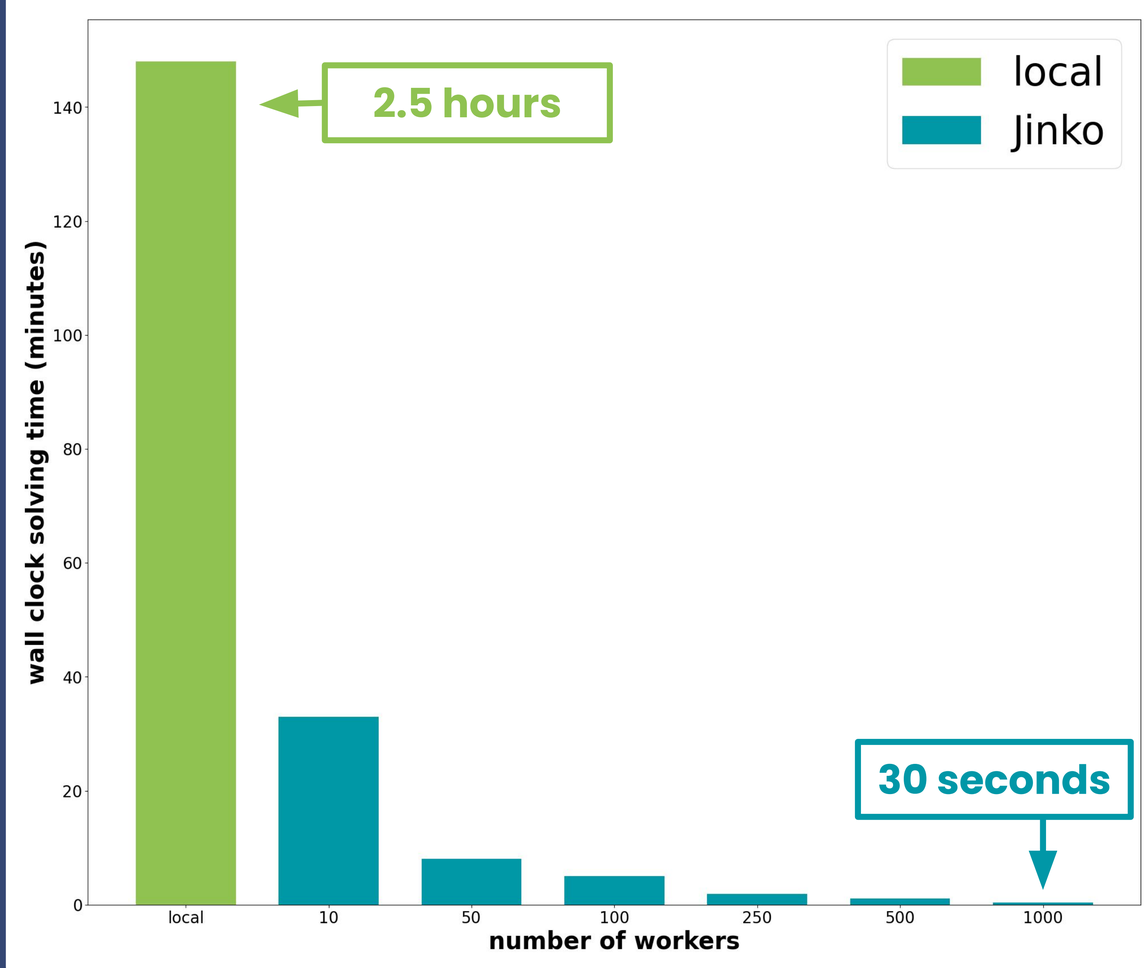

Figure 3: Cloud computing accelerates the solving of in silico clinical trials

BACKGROUND

Jinkō.ai integrated cloud solving solution leverages **cloud** computing, **parallelism**, just-in-time (JIT) compilation[1] and a state-of-the art ODE solver[2] to quickly simulate in silico trials.

METHODS

Just In Time Compilation: Jinkō is a generic simulation platform: it can solve any model created, edited or imported into it, from different file formats, including SBML. The model is stored in a symbolic way, allowing information extraction, transformations and solving. During the solving, the model equations must be evaluated multiple times. They can be interpreted by a generic function working on the symbolic representation (see "Eval" function in **Figure 2**). However, this approach, albeit flexible, is not efficient. In jinkō, we use Just In Time Compilation (JIT), to transform the symbolic representation to efficient machine code.

Scalable parallelism: Each patient in a virtual population is scheduled for solving for each of the protocol arms by being pushed in a job queue. (see **Figure 1**). The scheduler then registers virtual machines, workers, each will be responsible for solving one or many patients. The platform can virtually scale to any number of workers. We are using AWS spot instances in order to reduce costs.

We compared the wall clock time of running a virtual clinical trial in different scenarios:
1. With and without JIT compilation. The JIT compilation turns the symbolic computational model into fast machine code.
2. By varying the number of available remote workers.

These measurements were done in a "real life" scenario, where other simulations were running at the same time and measurement was done with a hand watch from start to finish of the simulation process. This is on purpose to explicit the kind of speedup which can be expected by using the jinkō platform.

RESULTS

When comparing solving times with and without **JIT**, we observe a **speedup** of **100x** by enabling JIT compilation. This does not take into account the time required for the JIT process.

For comparison, solving 30K patients on a developer laptop took 2h28, when it can be done in **under 30s** on jinkō.ai thanks to the platform scalability and parallelism over 1000 workers (see **Figure 3**). We observe a diminishing return when increasing the number of workers, this is due to the overhead of the parallelization, underutilization of the workers and delays in farm scaling due to AWS spot instances usage.

CONCLUSION & FUTURE WORK

The integration of JIT compilation, scalable cloud computing, and advanced ODE solving techniques significantly enhances the performance of clinical trial simulations. These innovations facilitate rapid, accurate analyses, thereby accelerating the drug development process and enabling more efficient clinical trial designs.

The platform is more efficient when handling "medium" sized models (models solved in a few seconds with JIT). When models are too fast to solve (less than a second per patient), the scheduling overhead dominates the total runtime. On the other hand, when models take minutes to solve, we achieve a close to perfect acceleration factor. The choice of AWS spot instances is relevant in our context, but on large models, the instances can be killed mid-solving, leading to a waste of time.

We are working toward a better scheduling strategy for "fast" models and to implement a resume mechanism for partially solved models in order to work around these limitations. Finally, we continuously improve the performance of the solver, including strategies to generate more efficient code to handle models with a lot of duplication (for example a cancer model with multiple tumors).

REFERENCES

[1] Lattner, Chris, and Vikram Adve. "The llvm compiler framework and infrastructure tutorial." International Workshop on Languages and Compilers for Parallel Computing (2004)
[2] Hindmarsh, Alan C., et al. "SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers." ACM Transactions on Mathematical Software (2005)