Fluid Limits for Formal Models of Software Performance

Mirco Tribastone
IMT – Institute for Advanced Studies Lucca

YEQT IX, Eindhoven
11 November 2015

Main Objective

How to analyse the performance of software intensive systems

Main Problem

How to scale programming languages for distributed systems that execute Markov chains instead of code

(Some queuing networks are specific programs of this language)

Programming Patterns

Master	Worker A	Worker B
`fork`	`fork`	`fork`
`join`	`computeA`	`computeB`
	`join`	`save`
		`join`

Red instructions: inter-process signal
Black instructions: local computation

Programming Patterns

Master	Worker A	Worker B
fork	*fork*	*fork*
join	computeA	computeB
	join	save
		join

Red instructions: inter-process signal
Black instructions: local computation

Programming Patterns

Master	Worker A	Worker B
`fork`	`fork`	`fork`
`join`	***`computeA`***	***`computeB`***
	`join`	`save`
		`join`

Red instructions: inter-process signal
Black instructions: local computation

Programming Patterns

Master	Worker A	Worker B
`fork`	`fork`	`fork`
`join`	`computeA`	***`computeB`***
	`join`	`save`
		`join`

Red instructions: inter-process signal
Black instructions: local computation

Programming Patterns

Master	Worker A	Worker B
`fork`	`fork`	`fork`
`join`	`computeA`	`computeB`
	`join`	***`save`***
		`join`

Red instructions: inter-process signal
Black instructions: local computation

Programming Patterns

Master	Worker A	Worker B
`fork`	`fork`	`fork`
`join`	`computeA`	`computeB`
	`join`	`save`
		`join`

Red instructions: inter-process signal
Black instructions: local computation

Programming Patterns

Master	Worker A	Worker B
`fork`	`fork`	`fork`
`join`	`computeA`	`computeB`
	`join`	`save`
		`join`

Red instructions: inter-process signal
Black instructions: local computation

Quantitative Semantics

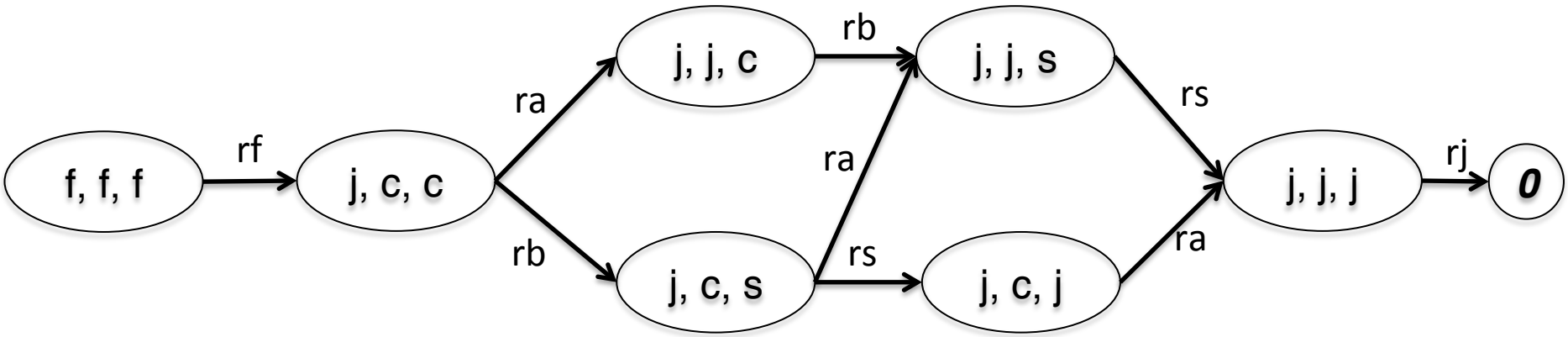
Label instructions with rates
(to model duration and data dependencies)

Master	Worker A	Worker B
`fork,rf`	`fork,rf`	`fork,rf`
`join,rj`	`computeA,ra`	`computeB,rb`
	`join,rf`	`save,rs`
		`join,rj`

Red instructions: inter-process signal
Black instructions: local computation

Quantitative Semantics

Compile program into a
continuous-time Markov chain



Master	Worker A	Worker B
`fork,rf`	`fork,rf`	`fork,rf`
`join,rj`	`computeA,ra`	`computeB,rb`
	`join,rf`	`save,rs`
		`join,rj`

Scaling Populations

Master[N_m]	**Worker A[N_a]**	**Worker B[N_b]**
`fork,rf`	`fork,rf`	`fork,rf`
`join,rj`	`computeA,ra`	`computeB,rb`
	`join,rj`	`save,rs`
		`join,rj`

- Create (identical) copies
  - A master can send signals to any worker
- The Markov chain **grows exponentially** with the population sizes

How to Cope

Master[N_m]	**Worker A[N_a]**	**Worker B[N_b]**
`fork,rf`	`fork,rf`	`fork,rf`
`join,rj`	`computeA,ra`	`computeB,rb`
	`join,rj`	`save,rs`
		`join,rj`

- Idea: *Symbolic* Semantics
  - Do not build the Markov chain
  - (Automatically) generate the possible transitions out of a generic state

Symbolic Semantics

Master[N_m]	Worker A[N_a]	Worker B[N_b]
`fork,rf`	`fork,rf`	`fork,rf`
`join,rj`	`computeA,ra`	`computeB,rb`
	`join,rj`	`save,rs`
		`join,rj`

1. New state representation:

$$\vec{X} = \left(X_{m,f}, X_{m,j}, X_{a,f}, X_{a,c}, X_{a,j}, X_{b,f}, X_{b,c}, X_{b,s}, X_{b,j} \right)$$

Counts how many processes are in each *local* state

Symbolic Semantics

Master[N_m]	Worker A[N_a]	Worker B[N_b]
fork,rf −	fork,rf −	fork,rf −
join,rj +	computeA,ra +	computeB,rb +
	join,rj	save,rs
		join,rj

2. Enumerate all transitions, e.g.

$$\vec{X} \rightarrow \vec{X} - \vec{1}_{m,f} - \vec{1}_{a,f} - \vec{1}_{b,f} + \vec{1}_{m,j} + \vec{1}_{a,c} + \vec{1}_{b,c}$$

at rate $r_f \min\left(X_{m,f}, X_{a,f}, X_{b,f}\right)$

Symbolic Semantics

Master[N_m]	**Worker A[N_a]**	**Worker B[N_b]**
`fork,rf`	`fork,rf`	`fork,rf`
`join,rj`	`computeA,ra`	`computeB,rb`
	`join,rj`	`save,rs`
		`join,rj`

2. Enumerate all transitions, e.g.

$$\vec{X} \rightarrow \vec{X} - \vec{1}_{a,c} + \vec{1}_{a,j}$$

at rate $r_a X_{a,c}$

Symbolic Semantics

Master[N_m]	**Worker A[N_a]**	**Worker B[N_b]**
`fork,rf`	`fork,rf`	`fork,rf`
`join,rj`	`computeA,ra`	`computeB,rb`
	`join,rj`	`save,rs`
		`join,rj`

3. Call $\vec{X}_k(t)$ the chain with $k \cdot N_m, k \cdot N_a, k \cdot N_b$ Masters, A-Workers and B-Workers.

   $\vec{X}_k(t)/k$ converges in the sense of Kurtz to an ODE limit.

ODE Limit

Master[N_m]

```
fork,rf
join,rj
```

Worker A[N_a]

```
fork,rf
computeA,ra
join,rj
```

Worker B[N_b]

```
fork,rf
computeB,rb
save,rs
join,rj
```

$$\frac{dx_{m,f}(t)}{dt} = -r_f \min\left\{x_{m,f}(t), x_{a,f}(t), x_{b,f}(t)\right\}$$

...

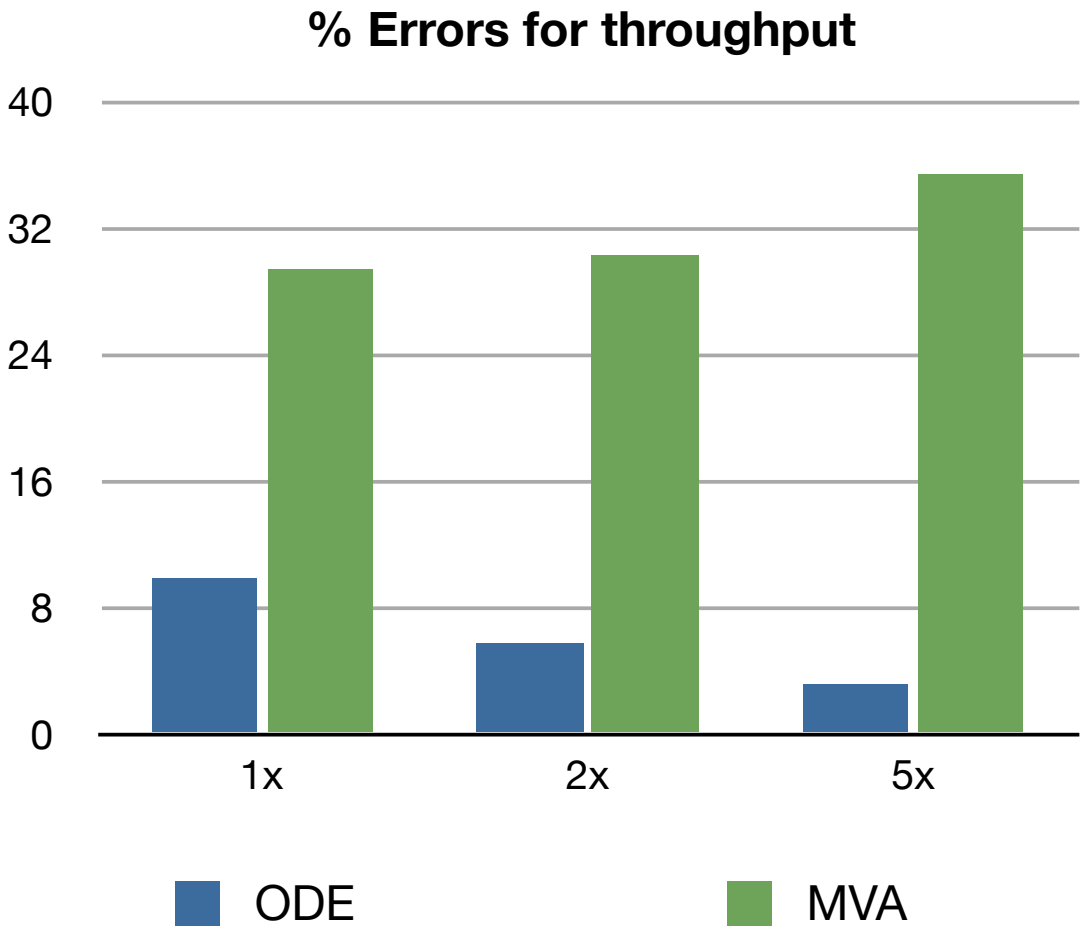
$$\frac{dx_{a,c}(t)}{dt} = -r_a x_{a,c}(t) + r_f \min\left\{x_{m,f}(t), x_{a,f}(t), x_{b,f}(t)\right\}$$

Properties

- The existence of a fluid limit is proven for the whole language
- In addition to fork/join it can model:
  - Simultaneous resource possession
  - Layered services
  - Services with vacation times
  - Asynchronous services
  - ...
  - Layered queuing networks!

[IEEE TSE'12, IEEE TSE'13]

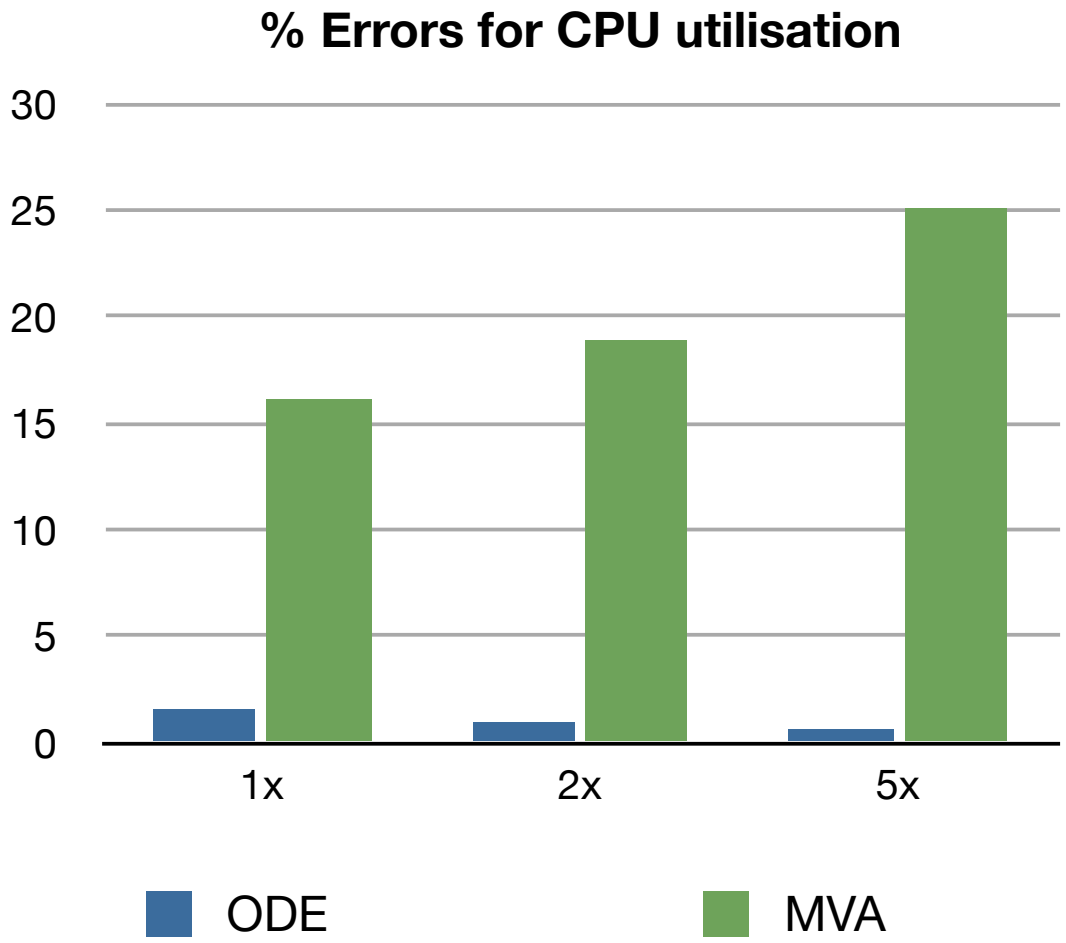
Comparison against LQNs



Errors with respect to Markov chain analysis as a function of the scaling factor

[IEEE TSE'13]

Comparison against LQNs



Errors with respect to Markov chain analysis as a function of the scaling factor

[IEEE TSE'13]

Summary

Master[N_m]	Worker A[N_a]	Worker B[N_b]
`fork,rf`	`fork,rf`	`fork,rf`
`join,rj`	`computeA,ra`	`computeB,rb`
	`join,rj`	`save,rs`
		`join,rj`

$$\vec{X} = \left(X_{m,f}, X_{m,j}, X_{a,f}, X_{a,c}, X_{a,j}, X_{b,f}, X_{b,c}, X_{b,s}, X_{b,j} \right)$$

- This representation is convenient when there are few distinct process types and/or few instructions per process type

Limitations

- In large-scale distributed systems there are **many** kinds of processes
  - **ODE Lumping techniques** [DSN'13, MFCS'15, TAC'15]
- Fluid limits as **first-order** approximations are inaccurate in certain workload regimes (near saturation)
  - **Moment-closure** approximations
  - How to justify theoretically?
- Fluid limits are **insensitive** to higher-order moments
  - How to model **non-exponential service times**

Bibliography:

- M. Tribastone, S. Gilmore, and J. Hillston, *"Scalable Differential Analysis of Process Algebra Models,"* **IEEE Transactions on Software Engineering**, 2012.
- M. Tribastone, S. Gilmore, and J. Hillston, *"Fluid Rewards for a Stochastic Process Algebra,"* **IEEE Transactions on Software Engineering**, 2012.
- G. Iacobelli and M. Tribastone, "Lumpability of fluid models with heterogeneous agent types", **DSN'13**.
- M. Tribastone, *"A Fluid Model for Layered Queuing Networks,"* **IEEE Transactions on Software Engineering**, 2013.
- G. Iacobelli, M. Tribastone, and A. Vandin, *"Differential Bisimulation for a Markovian Process Algebra,"* **MFCS'15**.
- M. Tschaikowski and M. Tribastone, *"Approximate reduction of heterogeneous nonlinear models with differential hulls"*, **IEEE Transactions on Automatic Control** (to appear).