Responsive, resilient, elastic and message driven system

solving scalability problems of course registrations

Janina Mincer-Daszkiewicz,

University of Warsaw

jmd@mimuw.edu.pl

Dundee, 2015-06-14

MUCI

Agenda

- Registration for courses and classes in USOS – various models.
- How to organize it better?
- Can reactive approach solve the problem?
- Technical aspects.
- Testing and going live.

USOS stands for University Study-Oriented System
(used by over 45 HEIs in Poland)

MUCI

Stating the problem

- Course registration is one of the most demanding functionalities of a student management system.
  - Logistics of the process
    - Short time deadlines.
    - Groups of stakeholders involved (authorities, administration, students, academic teachers).
  - Scale of the process
    - Medium size unit like Faculty of Mathematics, Informatics , Mechanics: 1500 students, 300 courses.
    - Large university like University of Warsaw: 26 units, 50 thousand students, largest faculty has 6 thousand students, 20 thousand courses.
- In USOS we cope with it since 2000 delivering support for various scenarious:
  - Some are more appealing for administration.
  - Others are more liked by students.
  - Some are not so much liked by university servers.
  - Those „likes” are contradictory.

Two-phase registration

- **First phase – choosing courses**
  - Students choose courses they would like to attend.
  - Decisions are made by authorities off-line based on the reports from the system and possibly some other criteria (e.g. grade averages).
  - There is time to either cancel the course or organize extra classes.
  - Students have to wait until the end of the phase to get the results.

- **Second phase - choosing classes (lectures, labs, project groups)**
  - The group registration module is switched on.
  - Students accepted for courses state their preferences.
  - Module is switched off.
  - Group registration engine assigns students to classes taking into account their preferences, trying to minimize the number of time conflicts and preserving other requirements (sophisticated algorithms are used!).
- **Both phases**
  - Students do not have to be available during registration at the same time or at the same place.

Direct (combined) registration for courses and classes

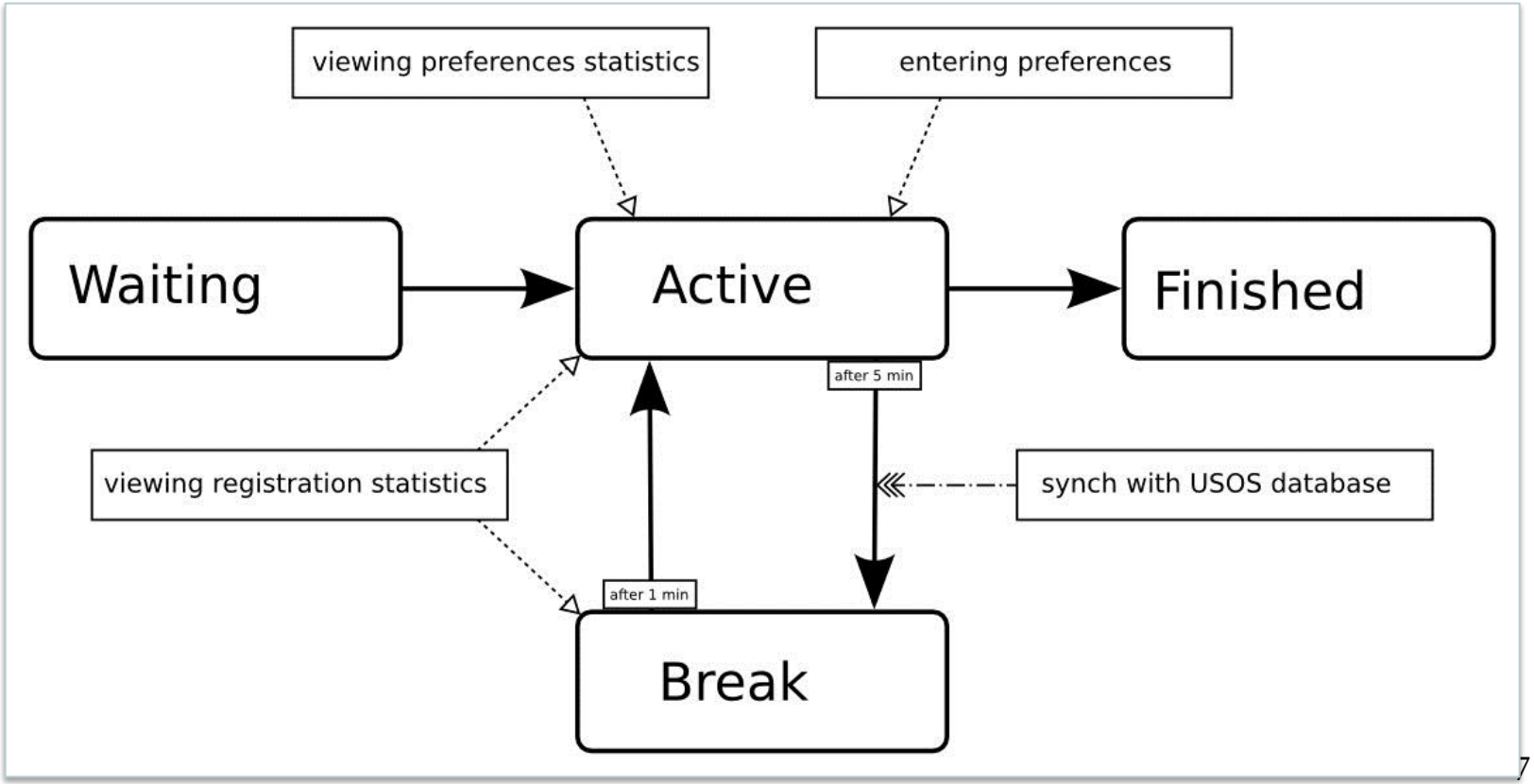
- More suited for courses offered centrally to all students of the university (like language courses or physical education classes).
- Liked by faculties which prefer less human involvement.
- Fully automatic.
- Students register directly for course groups on a **first come first served** basis.
- Students have to be **on-line** on the moment registration starts and **compete** with thousands of other students.
- Students get **immediate feedback**.
- System controls group limits, time deadlines and other requirements (courses may be dedicated to some groups of students).
- Variant with tokens (virtual money).
- **Stress-testing** for university servers.

How to organize it better?

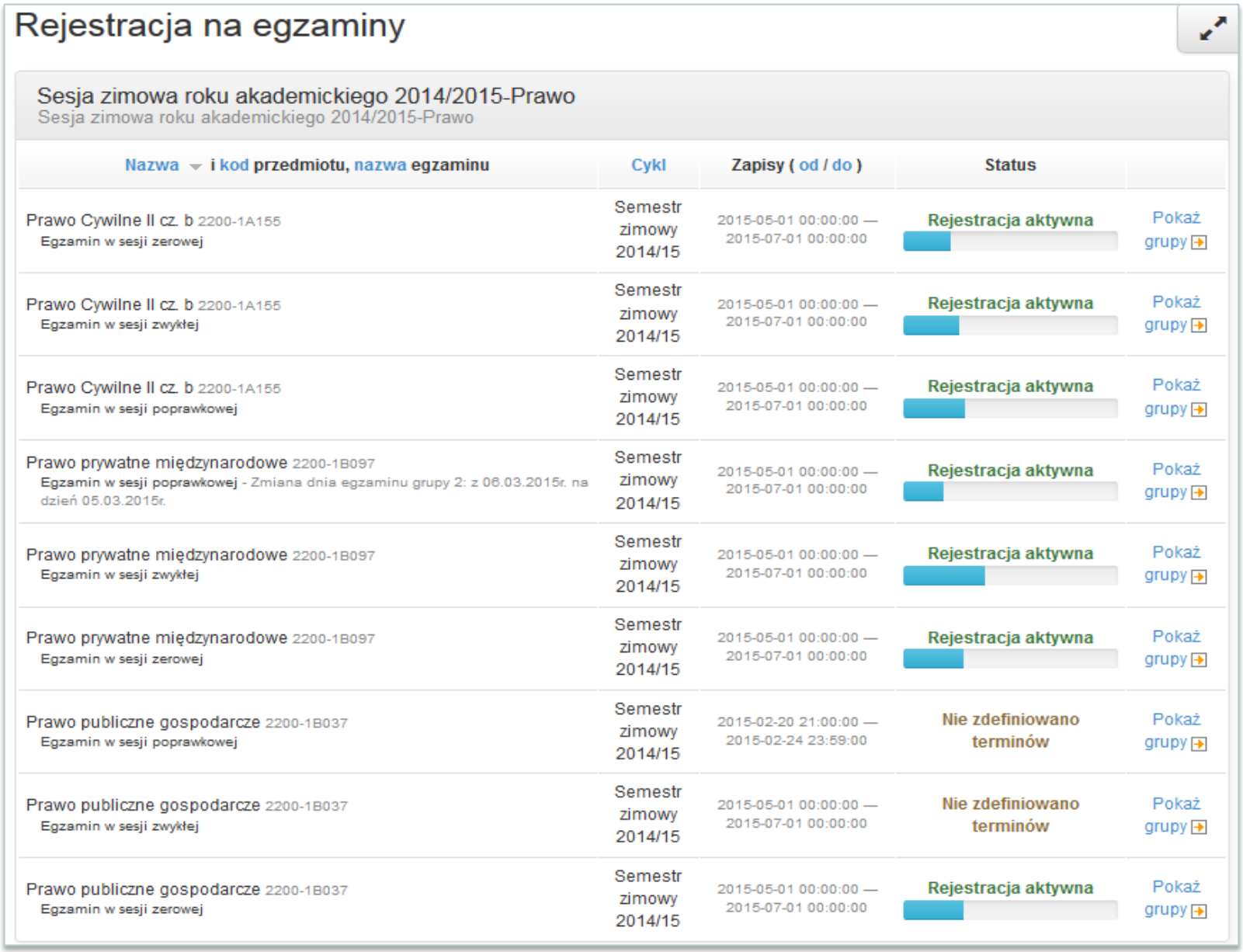
- **Direct registrations** are more appealing to students and administration giving immediate feedback and being fully automatic.
- **Two phase registrations** based on preferences are more user-oriented by giving the possibility to match supply and demand and also by being less demanding with respect to computing power.
- The optimal solution should stay user friendly but also get burden off the university administration, meaning both student's offices and IT departments.
- Solution – **micro rounds** (time periods lasting app. 5-10 min):
  - The registration starts with the first active micro round.
  - Students enter their preferences using a dynamic single page web application.
  - During a subsequent time interval backend server makes registration decisions which are immediately displayed to the students.
  - Having access to all requests, the server may optimize distribution of places between students.
  - For some students the registration ends, those less lucky continue the game delivering new preferences in the next active micro round.

Registration may be in one of the following stages

- **Waiting** — registration has not started yet but details of courses and classes are available to students who can plan in advance their preferable schedules.
- **Active** — micro round is active, students may define preferences. The micro round is short but long enough to diminish the pick loads of first come first served approach.
- **Break** — interval between two active micro rounds, students' requests are processed, results of registrations are displayed to students and stored in USOS database.
- **Finished** — registration is finished



List of registrations, showing time flow



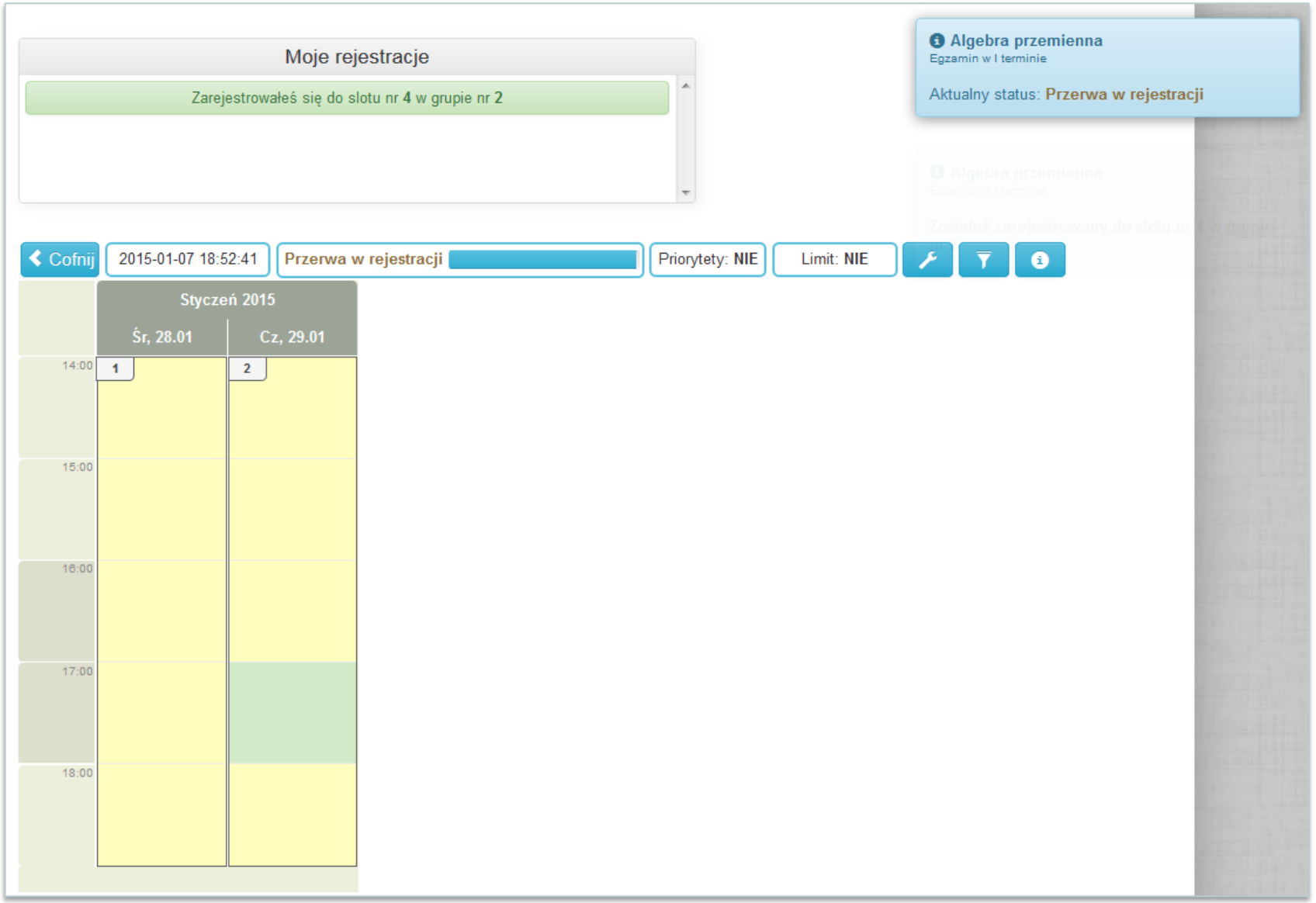
Rejestracja na egzaminy

Sesja zimowa roku akademickiego 2014/2015-Prawo

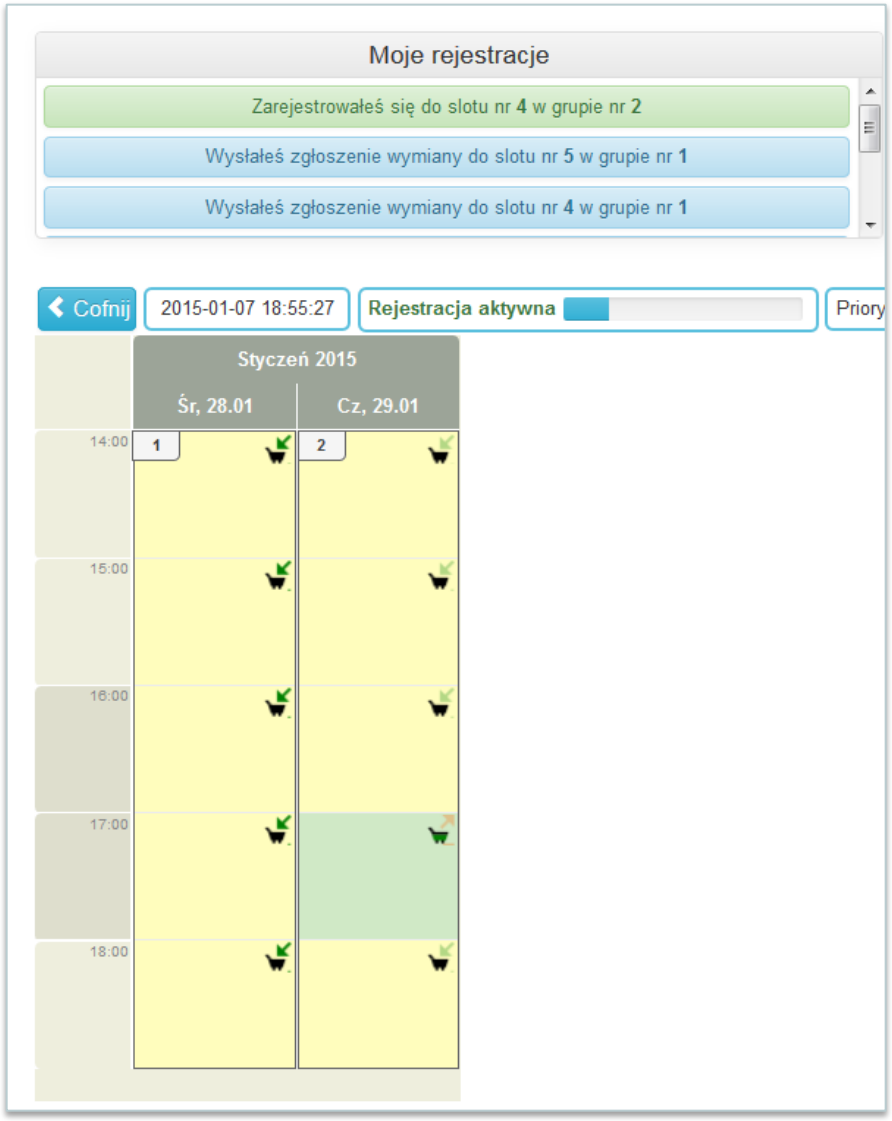
Sesja zimowa roku akademickiego 2014/2015-Prawo

Nazwa i kod przedmiotu, nazwa egzaminu	Cykl	Zapisy (od / do)	Status	
Prawo Cywilne II cz. b 2200-1A155 Egzamin w sesji zerowej	Semestr zimowy 2014/15	2015-05-01 00:00:00 — 2015-07-01 00:00:00	Rejestracja aktywna	Pokaż grupy
Prawo Cywilne II cz. b 2200-1A155 Egzamin w sesji zwykłej	Semestr zimowy 2014/15	2015-05-01 00:00:00 — 2015-07-01 00:00:00	Rejestracja aktywna	Pokaż grupy
Prawo Cywilne II cz. b 2200-1A155 Egzamin w sesji poprawkowej	Semestr zimowy 2014/15	2015-05-01 00:00:00 — 2015-07-01 00:00:00	Rejestracja aktywna	Pokaż grupy
Prawo prywatne międzynarodowe 2200-1B097 Egzamin w sesji poprawkowej - Zmiana dnia egzaminu grupy 2: z 06.03.2015r. na dzień 05.03.2015r.	Semestr zimowy 2014/15	2015-05-01 00:00:00 — 2015-07-01 00:00:00	Rejestracja aktywna	Pokaż grupy
Prawo prywatne międzynarodowe 2200-1B097 Egzamin w sesji zwykłej	Semestr zimowy 2014/15	2015-05-01 00:00:00 — 2015-07-01 00:00:00	Rejestracja aktywna	Pokaż grupy
Prawo prywatne międzynarodowe 2200-1B097 Egzamin w sesji zerowej	Semestr zimowy 2014/15	2015-05-01 00:00:00 — 2015-07-01 00:00:00	Rejestracja aktywna	Pokaż grupy
Prawo publiczne gospodarcze 2200-1B037 Egzamin w sesji poprawkowej	Semestr zimowy 2014/15	2015-02-20 21:00:00 — 2015-02-24 23:59:00	Nie zdefiniowano terminów	Pokaż grupy
Prawo publiczne gospodarcze 2200-1B037 Egzamin w sesji zwykłej	Semestr zimowy 2014/15	2015-05-01 00:00:00 — 2015-07-01 00:00:00	Nie zdefiniowano terminów	Pokaż grupy
Prawo publiczne gospodarcze 2200-1B037 Egzamin w sesji zerowej	Semestr zimowy 2014/15	2015-05-01 00:00:00 — 2015-07-01 00:00:00	Rejestracja aktywna	Pokaż grupy

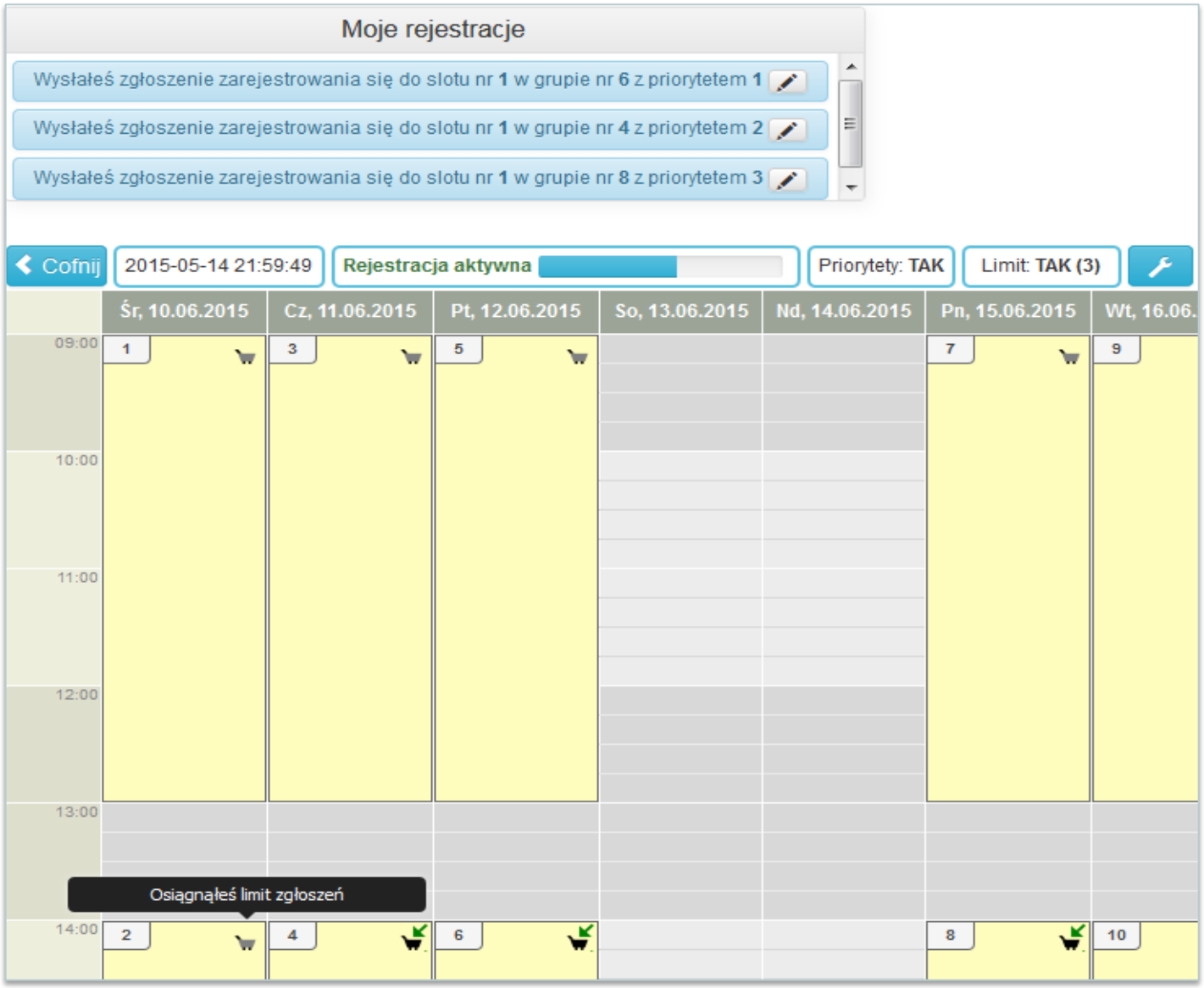
Schedule with two groups, each with 5 slots, break between micro rounds, messages appear dynamically on the right



Active micro rounds, pending requests are displayed on top, various baskets correspond to the various actions



More registration groups, each group with one slot, registration with priorities, some groups are blocked

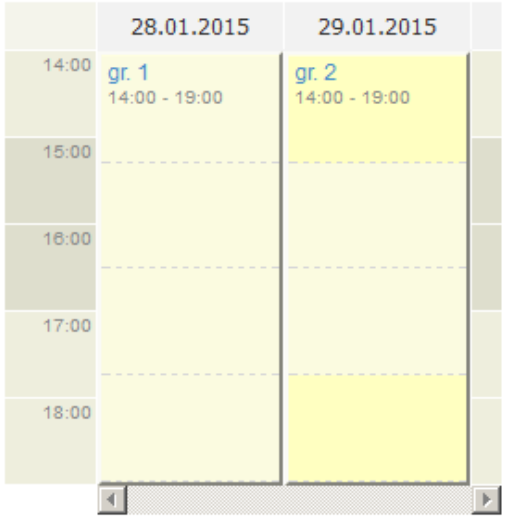


Lecturer can see registered students, send emails

Algebra przemienna 1000-1M14AP
Semestr zimowy 2014/15
Egzamin w I terminie EGZ1

Grupy egzaminacyjne

Pokaż grupy: **moje** **wszystkie**



Wyślij wiadomość do wszystkich studentów zapisanych na przedmiot

Wyślij wiadomość do wszystkich studentów zapisanych na egzamin

Numer grupy	Czas egzaminu	Miejsce	Zapełnienie miejsc	
1	**28.01.2015** 14:00 - 19:00	Budynek Dydaktyczny - Wydział Matematyki, Informatyki i Mechaniki - Kampus Ochota , pokój nr 4550	0 / 5	studenci
2	**29.01.2015** 14:00 - 19:00	Budynek Dydaktyczny - Wydział Matematyki, Informatyki i Mechaniki - Kampus Ochota , pokój nr 4550	3 / 5	studenci

MUCI

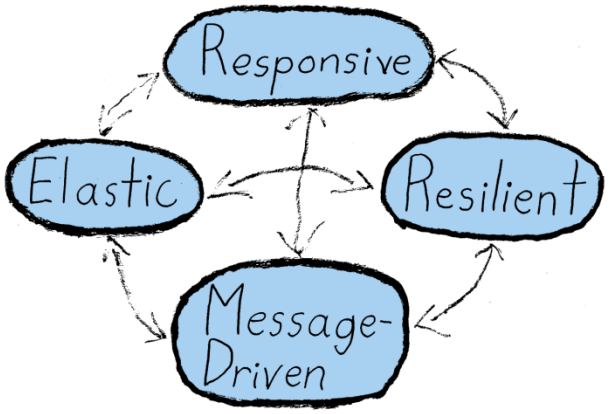
Reactive systems

Reactive manifesto

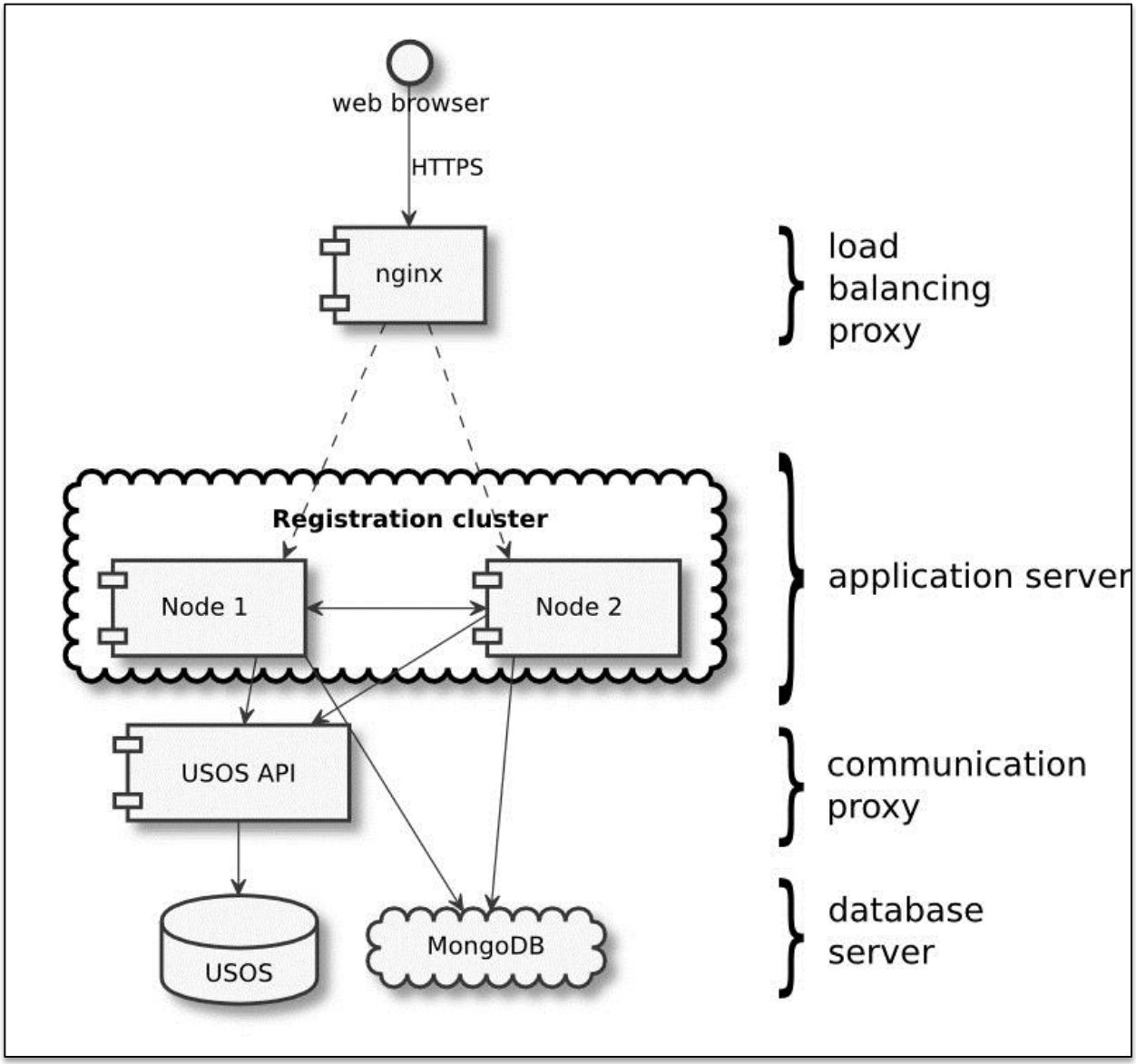
*Systems built as Reactive Systems are more **flexible**, **loosely-coupled** and **scalable**. This makes them easier to **develop** and amenable to **change**. They are significantly more tolerant of failure and when failure does occur they meet it with elegance rather than disaster. Reactive Systems are highly responsive, giving users effective interactive feedback.*

Requirements against the new model of registration comply with the principles of reactive applications — applications which are event driven.

By placing the new registration system in this context we can take advantage of the universal knowledge and experience, as well as patterns and ready solutions developed by the creation of similar systems.



General overview of the system architecture



MUCI

User interface

- User interface is not just a collection of static pages, but a separate web application.
- It has been written using **SPA** (**Single Page Application**) architecture.
- It runs in a browser, can immediately respond to a user action without sending to the server a request to render a new page.
- All HTML, JavaScript and CSS is taken during single page load or downloaded dynamically when necessary, usually in response to user actions.
- Downloaded resources are placed in the browser cache.
- Such applications can reduce the amount of data sent over the network and lower the load on the HTTP server.
- A classic example of such application is **Gmail**.
- We use client-side library **AngularJS** which is web browser JavaScript framework adopting SPA principles.

- The user running SPA has the impression as if it was a desktop application since the application almost immediately responds to user actions.

Load balancing proxy

- Application server is distributed to ensure proper performance and scalability.
- That means that we need a proxy responsible for load balancing.
- Mechanism used must comply with the following assumptions:
  - support for Server-Sent Events,
  - routing of calls from the same client to the same node — so called sticky sessions,
  - support for secure SSL connection,
  - support for proxy cache.
- **Nginx** has been chosen from various possible candidates.

Application server

- Application server is responsible for handling the registration process.
- It is implemented as a stand-alone application running in Java Virtual Machine.
- Data between user interface and the application servers and between application server and USOS API is sent by HTTP protocol in JSON format.
- It is a distributed **cluster of nodes**, from which everyone can accept any connections from customers.
- These can be both regular HTTP connections and asynchronous connections to send notifications.
- The nodes communicate between each other to support load balancing and to provide resistance to failure.
- Internal architecture of the server is based on the **model of cooperating actors.**
- Server is mostly written in **Scala** with some parts written in Java.
- We use **Akka** toolkit which supports the model of actor-based concurrency.

Communication proxy (USOS API)

- USOS is a suite of software applications built in a distributed architecture around a central Oracle database.
- **USOS API** is a standard REST-like interface to USOS, publicly available, well documented, with guaranteed backward compatibility.
- The application server uses USOS API to retrieve and store data from/to Oracle database.
- Results of methods are delivered in XML or JSON format.
- USOS API implements business logic and makes it available to other modules of the system.
- One HTTP request stores all served requests made for one course conducted at the end of each micro round. Thanks to this database is not unduly burden even when many students try to register in a short time.

Database server

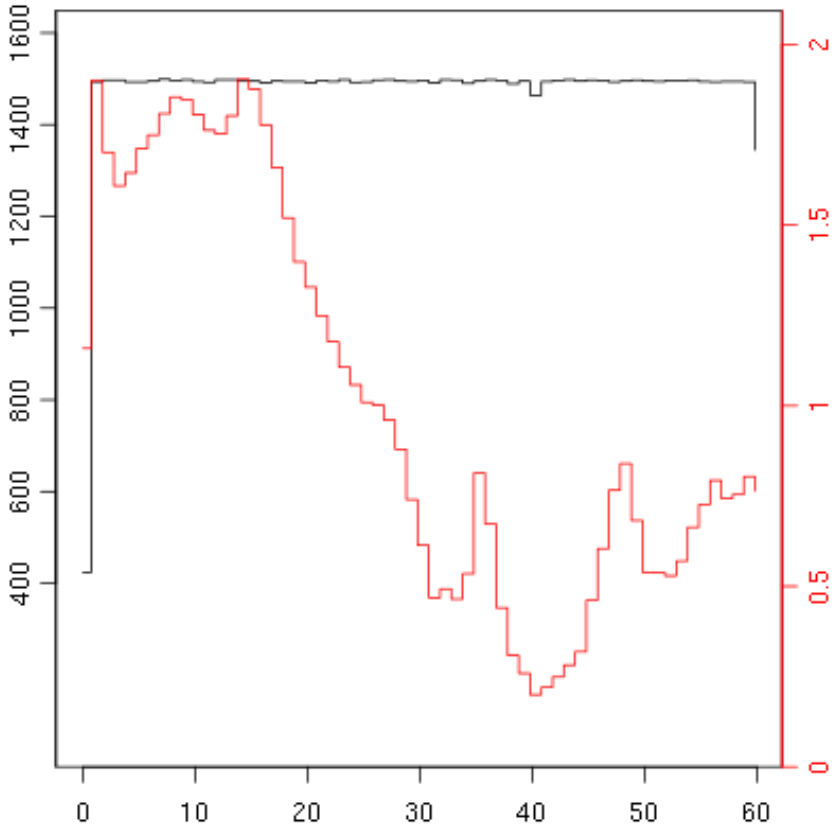
- Data associated with the application itself, such as sessions, registration requests, and provisionally collected data on courses and classes, are stored in a separate database.
- This database has to serve a large number of read and write requests, which can be processed in parallel (requests of different customers do not require any synchronization).
- **NoSQL databases** give the opportunity to achieve higher performance at the expense of deterioration of insulation of transactions, and are more cost efficient than commonly used relational databases.
- We have chosen **MongoDB**. Mongo nodes can be reproduced increasing the degree of data replication, and also their safety. Data can also be scattered between nodes, enabling greater degree of parallelization of read and write operations.
- The chosen solution allows to achieve two goals:
  - Take off the load from Oracle with minimal interference with the existing structure present in the database.
  - Use the local database MongoDB, so that several common operations performed by the students — registration requests — are processed locally, in an efficient manner.

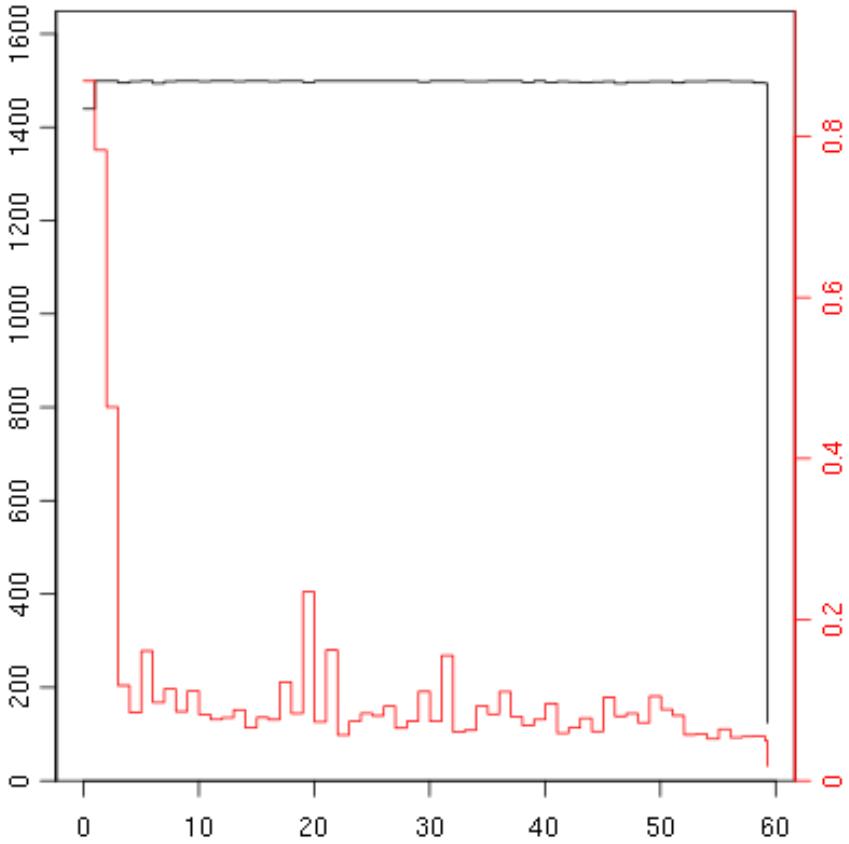
MUCI

Testing

- The preliminary tests were run in a computer lab of the Faculty of Mathematics, Informatics, and Mechanics of the University of Warsaw, on desktop class computers.
- Each machine hosted one MongoDB and one application server.
- Experiments were conducted for the increasing number of machines and increasing number of requests per second.
- We looked for the moment the system is not able to handle the increasing workload.
- Axis X shows the time of the experiment in seconds.
- Left (black) axis Y shows the overall number of requests initiated in that second.
- Right (red) axis Y shows the average response time for the request started in that second (*the smaller the better*).

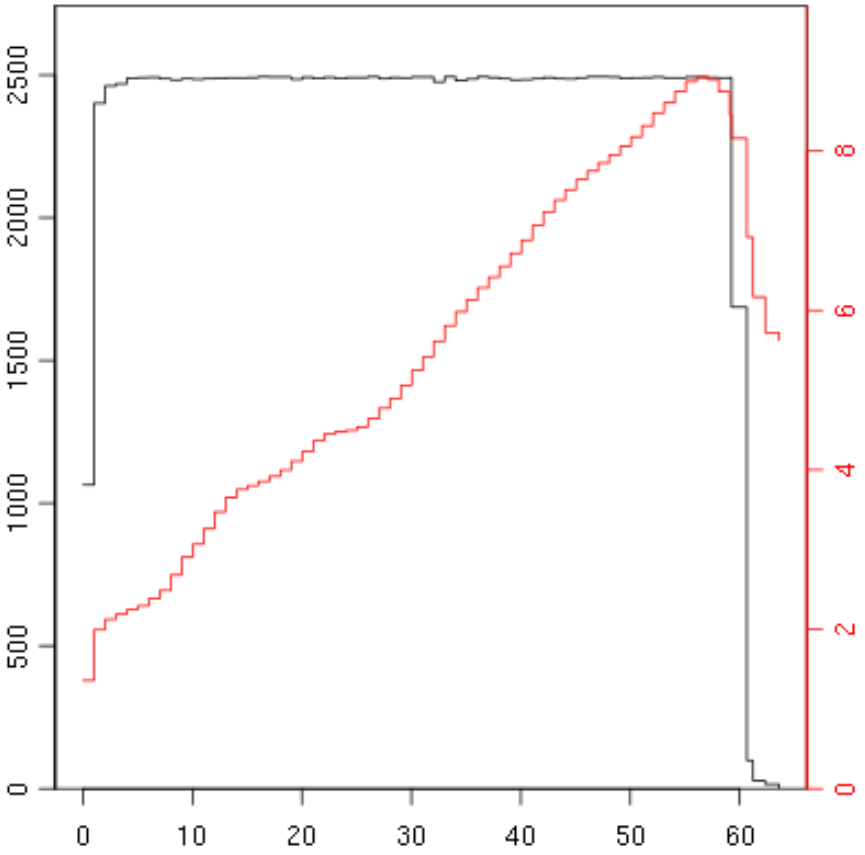
Test results for one machine (left) and two machines (right) for 1500 requests per second

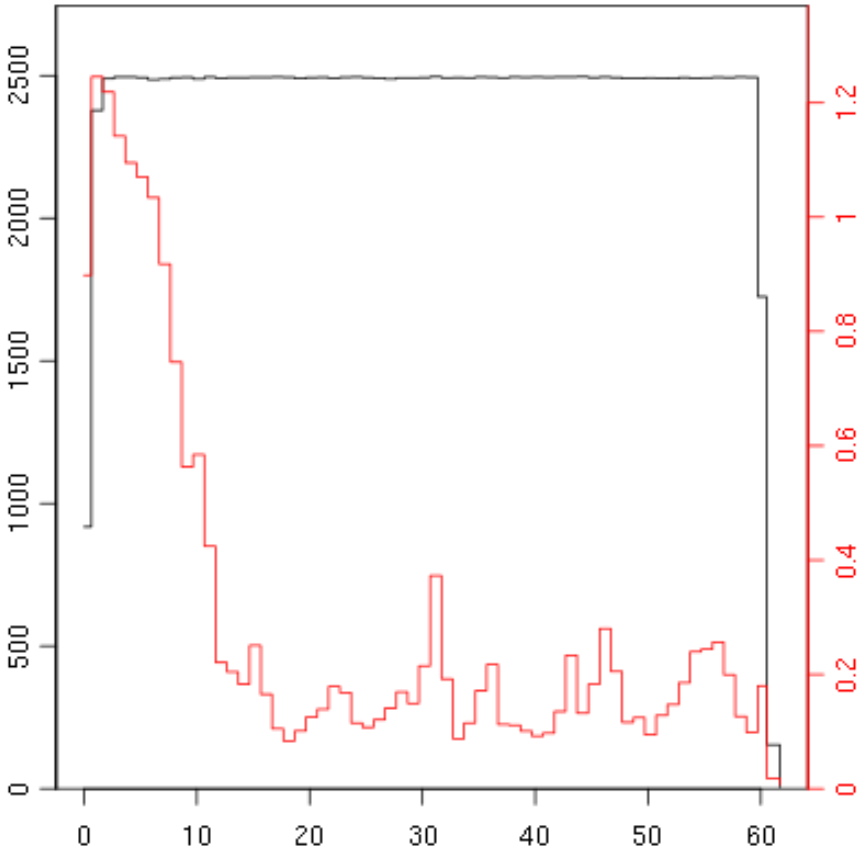




Adding the second machine we reduce the average response time to a value imperceptible to the human.

Test results for two machines (left) and three machines (right) for 2500 requests per second





System scales well - one more machine allows to lower the average response time to an acceptable value of less than 0,2 seconds.

MUCI

Conclusions – 1

- The model of **micro rounds** gets burden off the university administration, gives immediate feedback to students, avoids FCFS approach
- To achieve the respective **responsiveness**, **scalability** and **resilience**, involved technologies were chosen carefully.
- Backend server runs in asynchronous and distributed computation model of **cooperating actors** which exchange messages.
- Data is stored in **NoSQL** database kept in main memory for most of the time and the frontend is designed as a **dynamic single page web application**.
- Both the operation of the server and the user interface is **event-driven**.
- Through the use of modern programming techniques the new registration system reaches **high scalability** which makes it possible to increase throughput at key moments of the academic year.

MUCI

Conclusions – 2

- It is possible to **increase performance within the same machine** by increasing amount of available computing resources.
- In case of special requirements, it is possible to run the application on **multiple machines**.
- The end result is a registration system capable of handling 1500 requests per second on a single desktop class computer, while preserving **response time acceptable to the human**.
- With demonstrated scalability, performance on professional servers, as well as their clusters, will be significantly better.
- Behavior on multiple machines is essential to achieve reliability in case of failure of individual machines.
- First real life usage of the new registration system is planned for spring 2015 registrations.

Acknowledgments

This paper is based on

the **Master thesis** of **Grzegorz Swatowski, Maxymilian Śmiech, Michał Żak**

USOSregistration – scalable registration system

supervised by Janina Mincer-Daszkiewicz

Programming work was done by **Grzegorz, Max and Michał** under the guidance of a senior programmer **Michał Kurzydłowski**