Konzeption und Realisierung eines Frameworks für Verteiltes Rechnen in Webbrowsern mittels WebRTC

Master-Thesis

von

Matthieu Holzer

Übersicht

- Einleitung
- Grundlagen und Technologiestand
- Konzeption
- Implementierung
- Beispiele & Evaluation
- Resümee
- Demo

Problemstellung

- verteiltes Rechnen = Softwareinstallation
- kaum **plattformunabhängig**
- Schlechte Unterstützung für **mobile Geräte**
- Java oder C, **JavaScript** heute oft beliebter
- Projekte mit JavaScript-Ansatz:
  - beschränkt auf **spezifische** Problematik
  - zentralisiert → **Single Point of Failure**

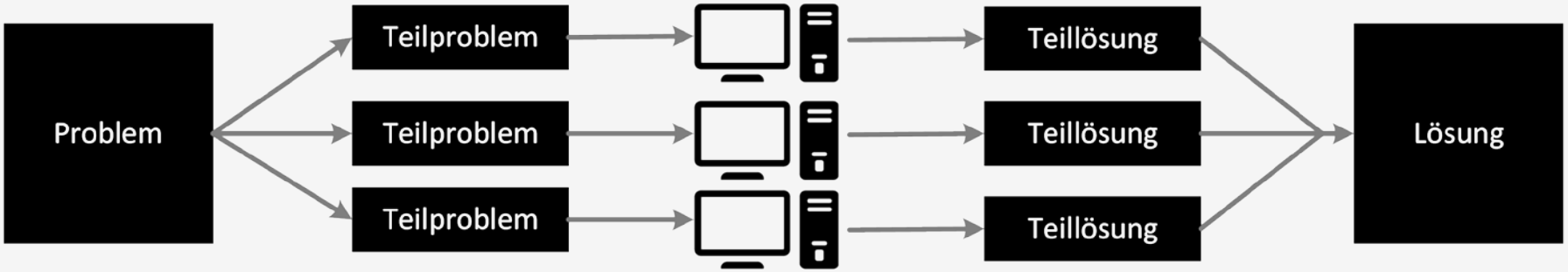
Anforderungen

- Einfachheit
- Flexibilität
- Plattformunabhängigkeit
- Stabilität & Reliabilität
- Sicherheit
- Skalierbarkeit
- Autonomie
- geringe Kosten

Zielsetzung

- Entwicklung eines prototypischen JavaScript-**Frameworks**
- **dezentralisiertes** System
- **browserbasiert**
- **plattformunabhängig**
- leicht **konfigurierbar**
- frei für Entwickler nutzbar (**Open Source**)

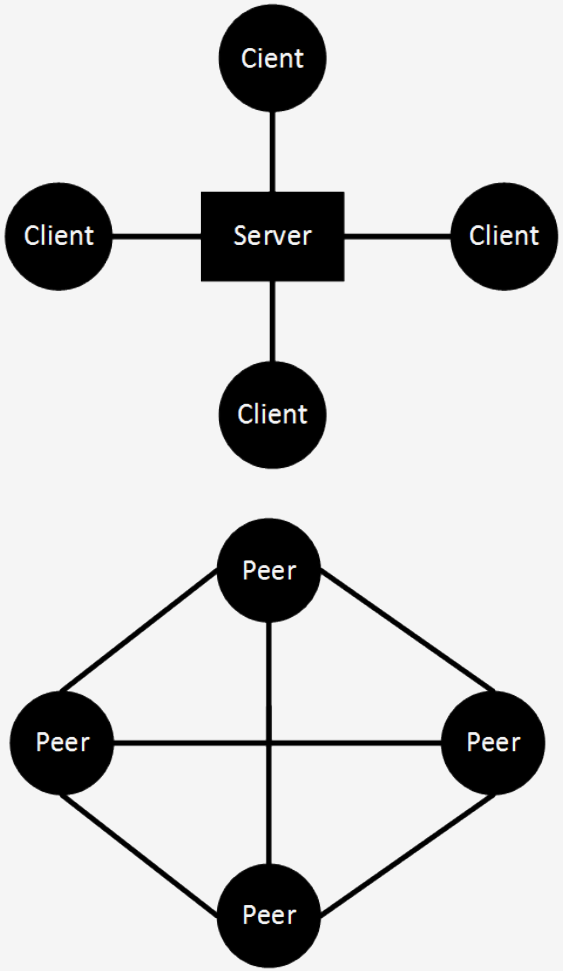
Verteiltes Rechnen - Distributed Computing



- Aufgabenverteilung auf Computer
- rechenintensive Projekte
- Problem muss *teilbar* sein

Peer-to-Peer (P2P)

- Netzwerk aus gleichgestellten Teilnehmern (Peers)
- direkte Kommunikation
- Struktur von P2P-Netzwerken
  - strukturiert / unstrukturiert
- Formen von P2P-Netzwerken:
  - zentralisiert, rein, hybrid



WebRTC

- JavaScript API
- P2P-Verbindung zwischen Browsern
- Echtzeitanwendungen (VoIP, Video, Chat, …)
- DataChannel API
- Transport: SCTP über UDP
- Verschlüsselung: DTLS
- Verbindung: ICE (STUN/TURN)
- Session: JSEP (SDP)

Weitere HTML 5 Techniken

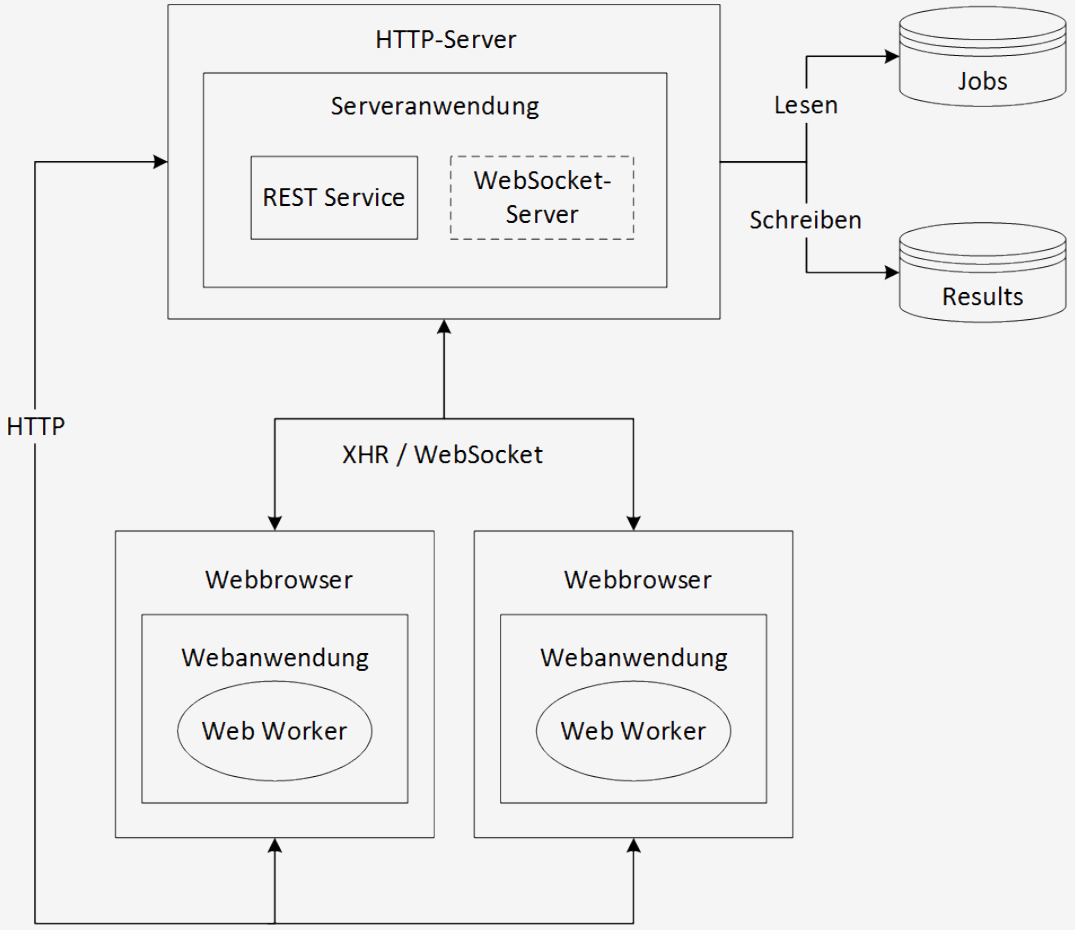
- **Web Worker** API → Threads
- **FileSystem** API → Dateisystem
- **Indexed Database** API → Datenbank
- **WebSocket** API → Signaling Channel
- **Geolocation** API → Lokalisierung
- **XMLHTTPRequest** → Hintergrundkommunikation

Relevante JavaScript-Projekte

CrowdProcess

Queen

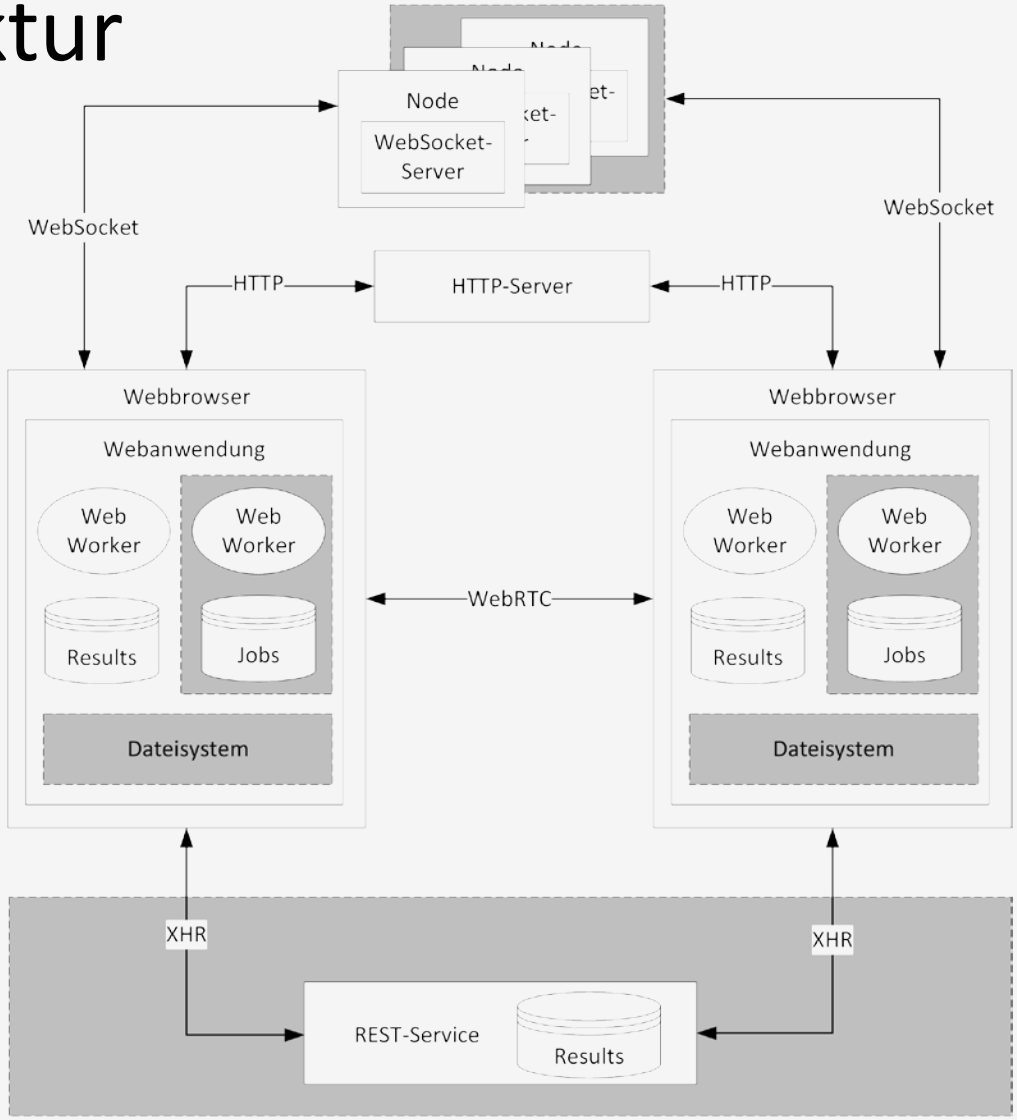
WeevilScout



Vorüberlegungen

- **Netzwerk**
  - Unstrukturiertes P2P-Netzwerk
  - Node als Signaling Service (ICE, SDP)
- **Berechnungsumgebung**
  - Worker- & optionaler Factory-Thread
  - isolierte externe Skripte (Sandbox)
  - Schnittstelle für Ressourcenzugriff
- **Speicherfunktion**
  - lokale Datenbank & Dateisystem
  - optionaler globaler Service via XHR

Architektur

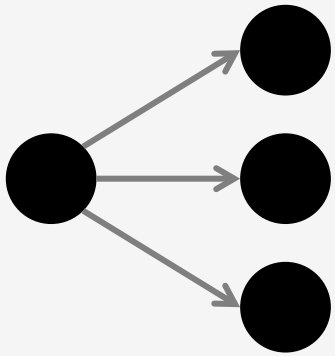


Ressourcen

- **Jobs** (Aufgaben) und **Results** (Lösungen)
- **Dateien** (Threads und Daten)
- **Synchronisierung**
  - beim Verbindungsaufbau zwischen Peers
  - nach definierbarem Intervall
- **Scheduling**
  - Prinzip: lock(x), solve(x), unlock(x)

Broadcast

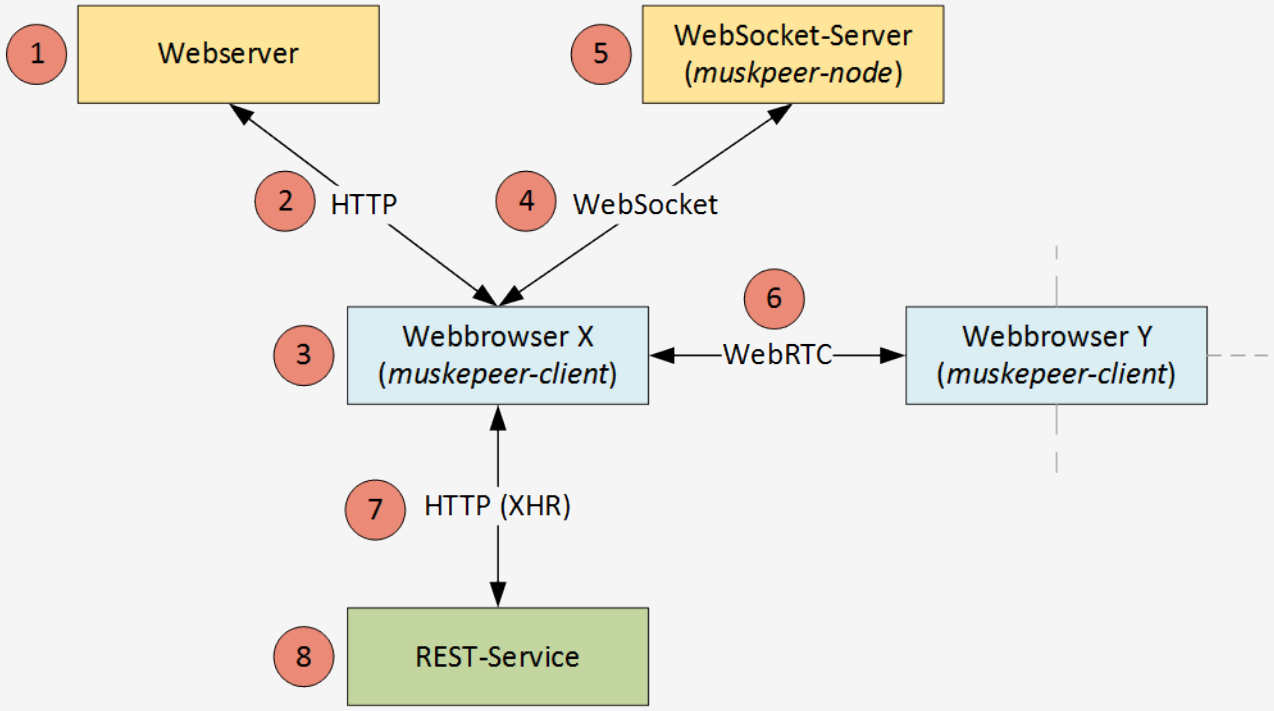
- Verbreitung von Nachrichten im P2P-Netz
- *Blind* oder *Simple* flooding
  - einfach zu realisieren
  - hohe Erfolgswahrscheinlichkeit
  - Redundanz → *Storm Problem*



- *V*ermeidung durch
  - Verzögerung (*Random Assessment Delay*)
  - Zeitstempel (*Time-to-Live*)

Sicherheit

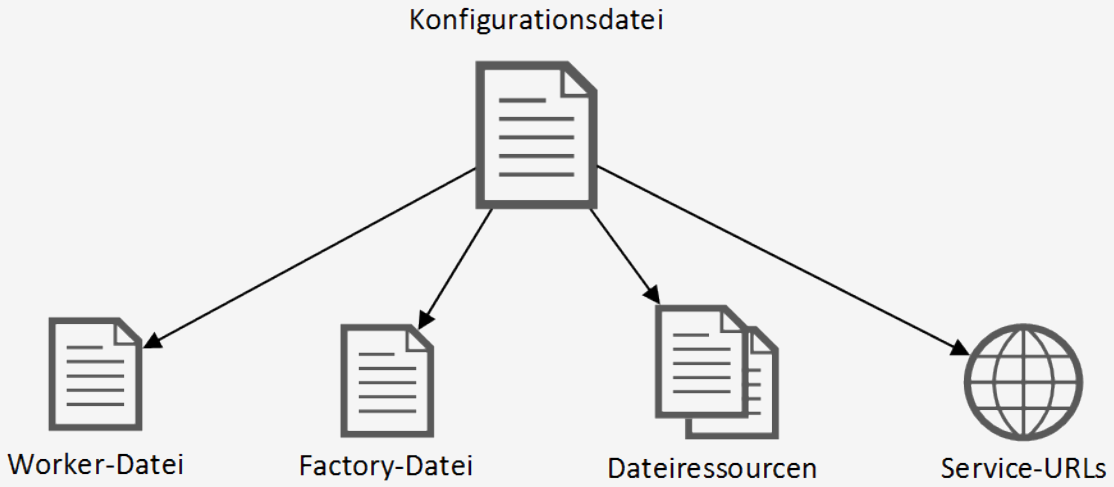
- Missbrauch (Botnetze, *Parasitic Computing*)
- Manipulation des Systems



Reliabilität

- Validierung der Daten
  - Umkehrfunktion
  - erneutes Berechnen (mehrere Iterationen)
- Stichproben & Reputationssystem
  - Peer erhält Bewertung durch andere Peers
  - schlechte Bewertung → Ausschuss
  - gute Bewertung → vertrauenswürdig
  - Problem: *Sybil-Attacke*

Konfiguration



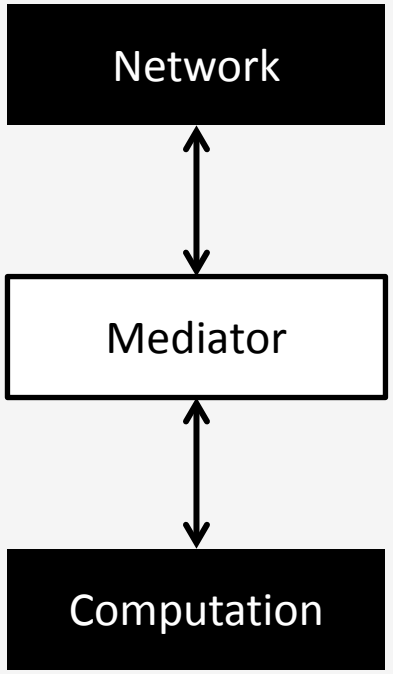
- Projektdatei als zentrale Informationsquelle
- ähnlich zu Torrent-Datei
- sämtliche Meta und Konfigurationsdaten
- einfache Weitergabe (*URL, Social Media, QR-Code*)

Kommunikation

- **Global (Peers & Nodes**)
  - JSON-Nachrichten über WebSocket bzw. DataChannel
  - Push/Pull-Prinzip
- **Lokal (Anwendung & Thread)**
  - JavaScript Objekte
  - Push/Pull-Prinzip
  - *Transferable Object* für Dateien (*zero-copy*)

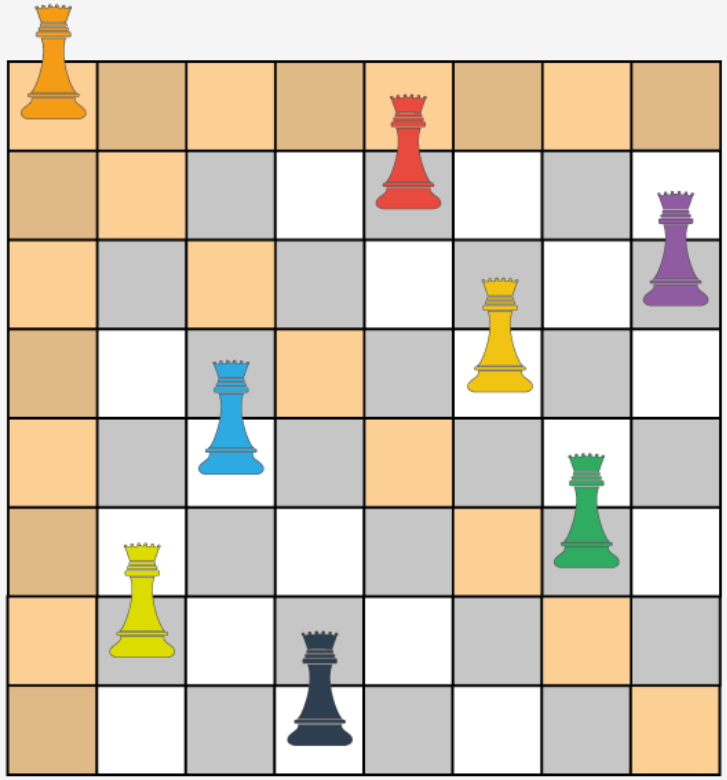
Besonderheiten

- Lose Kopplung der Module
  - Observer-Pattern
  - Mediator-Pattern
  - Events, Listener & Callbacks
- Asynchroner Ablauf
  - Promises für kontrollierten Ablauf
- Caching
  - IndexedDB zu langsam und blockierend

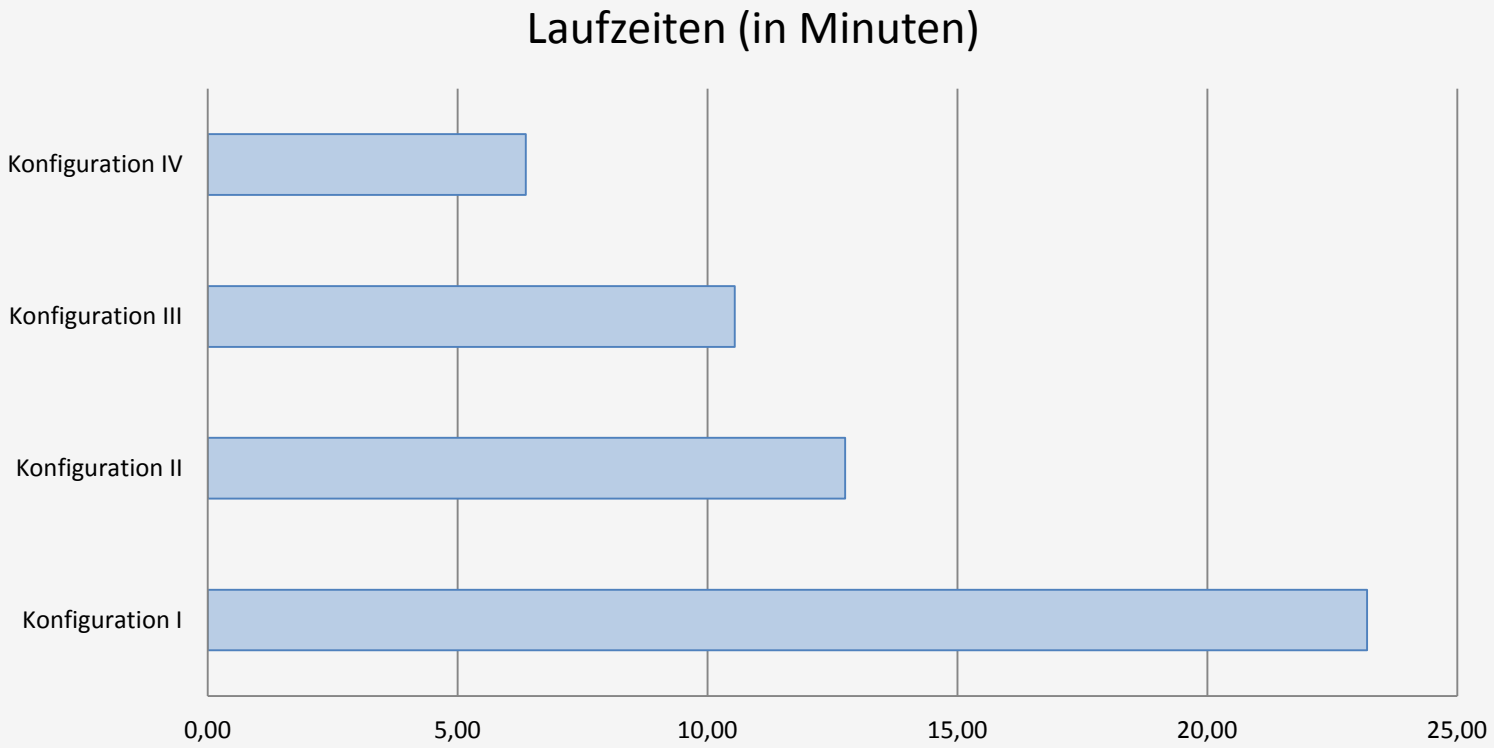


Beispiel I - Damenproblem

- NxN-Schachbrett
- Damen dürfen sich nicht „schlagen“
- Aufwand abhängig von N
- gut teilbar
- Lösungen bis N = 26 bekannt



Beispiel I - Damenproblem

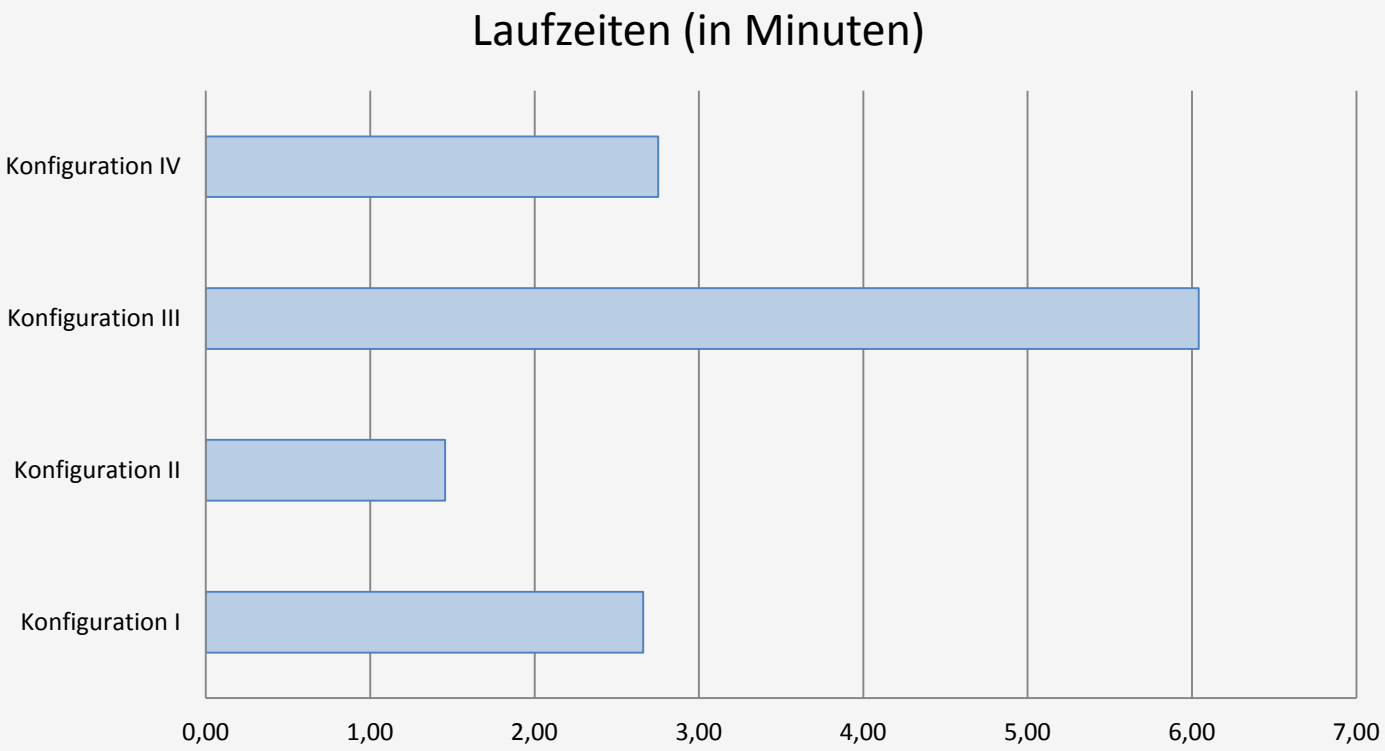


	Konfiguration I	Konfiguration II	Konfiguration III	Konfiguration IV
Median	23,198	12,750	10,543	6,363

Beispiel II - k-Nearest-Neighbor

- Klassifizierung von Datensätzen
- Test- und Trainingsdatensätze
- Abstandsfunktion
- Schlussfolgerung durch Nachbarklassifikation
- Gewichtung nach Ähnlichkeit möglich (*Kernel*)
- Visualisierung: http://bit.ly/1jDSDr9

Beispiel II - k-Nearest-Neighbor



	Konfiguration I	Konfiguration II	Konfiguration III	Konfiguration IV
Median	2,662	1,455	6,041	2,753

Beispiel II - Auswertung

- zunächst Leistungssteigerung
- Mehr als zwei Teilnehmer:
  - Fehler bei Peers → teilweise Absturz
  - Broadcast-Problem
  - Zu viele Nachrichten
  - Broadcast benötigt Optimierung

Ausblick

- Keine Begrenzung der Leistungsaufnahme
- JavaScript wird performanter
- GPGPU-Computing (WebGL, WebCL)
- Projekt frei auf GitHub verfügbar
  unter MIT-Lizenz
- GUI in Bearbeitung
- Abzuwarten wie WebRTC etc. sich entwickelt

Fazit

- Verteiltes *dezentrales* Rechnen im Browser möglich
- Optimierungen notwenig
  (*Broadcast, Dateiübertragungen*)
- Puffer-Speicher-Problematik
- Mehr Tests mit mehr Teilnehmern nötig
- mobile Geräte *noch* nicht reif?

Demo

https://muskepeer.net