Universität Hamburg
Fachbereich Informatik
Fachbereich oder #Student oder so

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

[Arbeitstitel:]Task-Scheduling in verteilter Softwareentwicklung

Seminararbeit

Seminar: Konzepte verteilter Softwareentwicklung

Louis Kobras
Matr.Nr. 6658699
4kobras@informatik.uni-hamburg.de

3. Juni 2016

Abstract

Suspendisse eu nunc. Aliquam dignissim urna sit amet mauris. Cras commodo, urna ut porttitor venenatis, arcu metus sodales risus, vitae gravida sapien ligula in est. Donec vulputate sollicitudin wisi. Donec vehicula, est id interdum ornare, nibh tellus consectetuer justo, a ultrices felis erat at lectus. In est massa, malesuada non, suscipit at, ullamcorper eu, elit. Nam nulla lacus, bibendum sit amet, sagittis sed, tempor eget, libero. Praesent ligula. Suspendisse nulla. Etiam diam. Nulla ante diam, vestibulum et, aliquet ac, imperdiet vitae, urna. Fusce tincidunt lacus vel elit. Maecenas dictum, tortor non euismod bibendum, pede nibh pretium tellus, at dignissim leo eros eget pede. Nulla venenatis eleifend eros. Aenean ut odio dignissim augue rutrum faucibus. Fusce posuere, tellus eget viverra mattis, erat tellus porta mi, at facilisis sem nibh non urna. Phasellus quis turpis quis mauris suscipit vulputate. Sed interdum lacus non velit. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae;

Inhaltsverzeichnis

1 Motivation

Scheduling ist ein kritischer Punkt in großen (Entwicklungs-)Projekten. Besteht ein Entwicklerteam aus nur wenigen Leuten, so lassen sich mit relativ geringem Aufwand Punkte finden, an denen sich etwa eine Teambesprechung anbietet, oder eine Deadline festlegen, wann eine Ressource wieder freigegeben werden muss. Mit wachsender Projektgröße steigt die Anzahl an Anforderungen jedoch immens.

Jede einzelne Person hat private Termine, die eingehalten werden müssen; Teilgruppen innerhalb des Teams müssen sich absprechen; der Kunde will betreut werden; Ressourcen müssen wieder freigegeben werden; Projektteile warten auf andere Projektteile.

All diese Bedingungen und Abhängigkeiten zu vereinen, ist eine zunehmend komplexe Aufgabe, je mehr das Team anwächst, je größer das Projekt wird.

Ziel dieser Arbeit ist es, mithilfe von Algorithmen aus der theoretischen Informatik ein Modell zu finden, welches all jenes miteinander vereint und in der Lage ist, einen optimierten Zeitplan zu bestimmen, der sämtliche ihm bekannten Termine und Abhängigkeiten berücksichtigt.

Dazu wird sich verschiedenen Problemen aus der NPH-Klasse bedient, deren Lösungsalgorithmen zwar eine lange Laufzeit haben, von denen sich jedoch einige zur Modellierung des hier gegebenen Problems anbieten. Zum Abschluss soll ein gefundenes Schedule evaluiert werden: Kann es aus Sicht des Teams optimiert werden, d.h. kann die gesamte Projektlaufzeit verkürzt werden? Kann es aus Sicht eines Agenten optimiert werden, d.h. ist das Schedule derart gestaltet, dass die Wartezeiten jedes einzelnen Agenten minimal sind? Hierzu wird sich einiger Grundkonzepte der Spieltheorie bedient.

2 Warum kein Dudel?

Dudel oder etwas Vergleichbares ist sicherlich vielen ein Begriff. Geht es darum, einen einzelnen Zeitslot zu finden, an dem möglichst viele Agenten zur Verfügung stehen, so ist Dudel durchaus eine überlegenswerte Lösung. Sowohl das Erstellen als auch das Abstimmen ist einfach in der Handhabung. Dennoch gibt es mehrere Gründe, warum für große Projekte Dudel ausscheidet.

Mangel an Übersichtlichkeit. Bei einem großen Projekt, welches von großem wirtschaftlichem Interesse für einen Kunden ist, kann es nötig sein, für einen Termin auf einen sehr feingliedrigen Zeitplan zurückzugreifen. Auch wenn die Erstellung, die Verwaltung und die Teilnahme an einem Dudel selbsterklärend sind, so nimmt die Komplexität zu, umso feingliedriger der Zeitplan sein soll. Ausgehend von einer Vollzeitwoche mit möglichen Zeitslots zu jeder Viertelstunde ergibt sich somit eine Zeitslotliste der Mächtigkeit $(8\text{h} \cdot 4 \text{ Slots pro Stunde}) \cdot 5 \text{ Tage } = 160$ Slots in einer Woche. Dies ergibt eine Matrix mit den Ausmaßen ($160 \times$Anzahl beteiligter Agenten). Bei einer derartigen Tabelle den Überblick zu behalten, sprengt den Rahmen realistischer Erwartungen an einen menschlichen Agenten in verwaltender Position.

Persönliche Motivation. Gegeben ein Arbeitstag von mehreren Stunden, ist es durchaus nicht undenkbar, dass ein Agent am Abend die Ressource Zeit, die ihm zur Verfügung steht, nicht freigibt, es sei denn, es ist erforderlich. Dies ist gerade bei einem Dudel leicht durchzuführen und

schwer nachzuvollziehen. Ein System, welches die Zeitpläne als Rohmaterial erhält, nimmt keine Rücksicht auf persönliche Abneigungen und eliminiert deswegen diesen Fall.

Nichtoptimierung. Ein Dudel zählt lediglich, wie viele Agenten zu welchen Zeitslots zur Verfügung stehen; es wird nicht weiter auf die Art der Blockierung der anderen Zeitslots eingegangen. Ein System, welches direkt mit Deadlines und Abhängigkeiten arbeitet, kann dazu im Stande sein, den Schedule derart zu berechnen, dass Wartezeiten reduziert werden, und ist deswegen potentiell in der Lage, einen optimierten Schedule zu erstellen, der die allgemeine Projektdauer reduziert und die Wartezeiten einzelner Agenten minimiert und somit die Produktivität maximiert.

Einzelläufigkeit. Ein Dudel hat nicht das Potenzial, nebenläufige Entwicklungsprozesse zu berücksichtigen. Es wird ein Zeitstrahl vorgegeben, auf dem maximierende Belegungen ausgegeben werden. Dem entgegen steht der Umstand, dass durchaus Sachen gleichzeitig geschehen können; so auch Termine, an denen Teilmengen der Agenten beteiligt sind. Für alle solche Ereignisse ein neues Dudel zu erstellen, ist umständlich und kann, da jeder Agent an mehreren Dudels teilnehmen muss, zu einer zeitlichen Belastung werden und somit die Produktivität senken.

Diese Gründe legen dar, warum die Dudel-Variante in großen Agentensystemen ungeeignet ist und ein eher dynamisch veranlagtes System zur Berechnung eines Schedules erforderlich ist.

3 Zerlegung in Teilprobleme

Da es leider den einen Superalgorithmus, der ein beliebig komplexes Problem in polynomieller Laufzeit löst, gibt, muss ein schwierigeres Problem in einfachere Teilprobleme aufgeteilt werden, die sich besser lösen lassen und für die möglicherweise schon eine Lösung bekannt ist. Zuerst wird die Fragestellung dreigeteilt in Eingabe, Rechnung und Ausgabe.

Die Eingabe nimmt die Zeiten eines jeden Akteurs und bestimmt daraus ein Format, mit dem weiter gearbeitet werden kann. Hierbei wird von allen Teilnehmern am Schedule erwartet, dass sie sämtliche Zeiträume eintragen, in denen sie nicht zur Verfügung stehen, sowie sämtliche Tasks, die sie betreffen, wobei der Projektleiter die Möglichkeit haben soll, die Projekt-Deadline anzugeben. Der Algorithmus soll dabei Duplikate erkennen und eliminieren können.

Diese Eingaben werden auf eine Form gebracht, die vom Algorithmus verarbeitet werden kann.

Anschließend werden diese verarbeiteten Eingaben sortiert, um einen Überblick darüber zu erhalten, wie ein Schedule aufgebaut ist. Hier kann bereits geprüft werden, ob es theoretisch ein Schedule geben kann, welches die Ramenbedingungen einhält, indem geprüft wird, ob ein einzelner Task mehr Ressourcen benötigt, als zur Verfügung stehen.

Die Rechnung wendet Lösungs- und Approximationsalgorithmen für bekannte und gelöste Probleme an, um die Aufgaben gleichmäßig zu verteilen, so dass ein Task von einer qualifizierten Entität bearbeitet wird, keine Entität gleichzeitig mehrere Tasks zu bearbeiten hat und keine übermäßigen Wartezeiten für untätige Entitäten entstehen. Dies impliziert eine größtmögliche Parallelisierung von wechselseitig unabhängigen Teilaufgaben.

Hierbei wird (nach der Sortierung in der Eingabe) darauf geachtet, dass alle Abhängigkeiten eines Tasks erfüllt sind, bevor er zugewiesen wird, und dass kein Task eine Deadline überschreitet.

Das System soll in der Lage sein, dynamisch im Hinblick auf zu erfüllende Abhängigkeiten eigene Deadlines für Meilensteine der Entwicklung festzulegen.

Die Ausgabe übernimmt den errechneten Schedule und erstellt daraus ein Diagramm, welches leicht lesbar und verständlich sein soll. Als Muster bieten sich Gantt-Diagramme oder Commit-Trees von Versionsverwaltungssystemen an.

4 Eingabe

```
//TODO: normierung
```

Es wird eine Reihe von Tasks erwartet, die allesamt eine (möglicherweise leere) Liste von Abhäng-igkeiten sowie optional eine Deadline besitzen. Diese Tasks werden mithilfe von TOPOSORT soriert.

Dazu folgende Reduktion:

Es wird ein gerichteter Graph konstruiert. Zunächst werden alle Task-Listen konkateniert. Diese Liste wird sortiert; Duplikate werden eliminiert.
Für jeden Task wird nun ein Knoten erzeugt; ist ein Task von einem anderen abhängig, so wird eine Kante entgegen der Richtung der Abhängigkeit zwischen den Knoten hinzugefügt. Auf diesem so entstandenen Graphen wird TOPOSORT durchgeführt.

Hinrichtung. Ist TOPOSORT erfolgreich, so gibt es keine Zyklen im Graphen, alle bekannten Abhängigkeiten können also aufgelöst werden.
Ist TOPOSORT nicht erfolgreich, so gibt es Zyklen im Graphen. Dies bedeutet, dass nicht alle Abhängigkeiten aufgelöst werden können, da einige Tasks indirekt von sich selber abhängen. In diesem Fall tritt eine Deadlock-Situation auf. Diese lässt sich dadurch beheben, einzelne Tasks zurückzuziehen, die einen Zyklus erzeugen. Andernfalls lässt sich kein vollständiges Schedule erzeugen.

Rückrichtung. Wenn es ein vollständiges Schedule gibt, so müssen alle Abhängigkeiten auflösbar sein. In diesem Fall lässt sich also ein Graph konstruieren, auf dem TOPOSORT erfolgreich terminiert.
Gibt es kein vollständiges Schedule aus dem Grund nicht aufgelöster Abhängigkeiten, so liegt dies daran, dass eine Abhängigkeit ewig auf Erfüllung wartet. Dies tritt nur bei Deadlock-Sitationen auf. Somit existiert im konstruierten Graphen ein Zyklus und TOPOSORT terminiert erfolglos.

5 Approximation durch NP-schwere Probleme

Zur Lösung eines Problems bietet es sich an, nach ähnlichen Problemen zu suchen, die bereits gelöst sind. In diesem Fall bieten sich folgende Probleme an:

- INTERVAL-SCHEDULING [1]
- MAX-SAT [2] / Knapsack
- LOAD BALANCING [3]

Algorithm 1 SCHEDULE(LIST TASKS, DATE DEADLINE)

```
1: for task in tasks do
2:     if task.timeRequirement is greater than deadline.isAway then
3:         return false
4:     end if
5: end for
6: Array A = TopoSort(tasks)
```

6 Lösungsansatz

```
//TODO: [sic]
```

7 Bewerten einer Lösung

```
//TODO: [sic]
```

[1] Definition vgl. INTERVAL-SCHEDULING
[2] Definition vgl. MAX-SAT
[3] Definition vgl. LOAD BALANCING

Definitionen

Nash-Gleichgewicht. (Spieltheorie) Eine Situation, in der alle beteiligten Agenten die Strategie aller anderen Agenten kennen und kein Agent die Motivation hat, einseitig von seiner Strategie abzuweichen

Pareto-Optimum. (Spieltheorie) Eine Situation, in der keine Partei ihre eigene Position verbessern kann, ohne dass sich die Position einer anderen Partei verschlechtert.

Auszahlungsoptimiert Eine Strategie ist für eine Entität auszahlungsmaximiert genau dann, wenn alle anderen Strategien, die diese Entität wählen kann, gleich viel oder weniger Gewinn bringen.

Eine Strategie ist für eine Gruppe auszahlungsmaximiert genau dann, wenn sie für alle Entitäten in der Gruppe auszahlungsmaximiert ist (also ein pareto-optimales Nash-Gleichgewicht vorliegt).

Task. Der Begriff *Task* wird hier als Oberbegriff für Aufgaben, Termine und Veranstaltungen verwendet.

Entität. Der Begriff *Entität* bezeichnet hier entweder einen Akteur oder eine Gruppe bzw. Partei.

Akteur. Oberbegriff für handende Menschen und Maschinen

Deadline.

a. Abschlusszeitpunkt eines Projektes

b. Zeitpunkt, zu dem ein Task spätestens erledigt sein muss, um den Schedule einzuhalten

LOAD BALANCING . (NP-schweres Problem nach [1])

Eingabe. m identische Maschinen, n Jobs, Job j besitzt die Ausführungsdauer t_j. Ein Job muss von einer einzelnen Maschine vollständig ohne Unterbrechung bearbeitet werden. Eine Maschine kann nur einen Job zur Zeit erledigen.

Frage. Gibt es eine Verteilung aller Jobs auf alle Maschinen derart, dass die Produktionsdauert minimiert ist?

INTERVAL-SCHEDULING .

```
//TODO: [sic]
```

(NP-vollständiges Problem nach [1])

Eingabe.

Frage.

MAX-SAT .

```
//TODO: [sic]
```

(NP-schweres Problem nach [1])

Eingabe.

Frage.

Projekt.

```
//TODO: PMBoK Definition
```

Literatur

[1] Thomas Andreae. Vertiefung kombinatorische optimierung. Vorlesungsfolien, 2016.

[2] Andreas Diekmann. *Spieltheorie: Einführung, Beispiele, Experimente*. Rowohlt Taschenbuch Verlag, April 2009 (4. Auflage).

[3] Project Management Institute. *A Guide to the Project Management Body of Knowledge*. 2013 (5. Auflage).