FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Autonom skalierendes, verteiltes Simulationsframework für TCAD-Anwendungen

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Harald Demel
Matrikelnummer 0728129

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Mag.rer.soc.oec. Dipl.-Ing. Dipl.-Ing. Dr.techn. Karl M. Göschka
Mitwirkung: Dipl.-Ing. Oskar Baumgartner, Dipl.-Ing. Markus Karner,
Dipl.-Ing. Christian Kernstock, Dipl.-Ing. Zlatan Stanojević

Wien, 06.07.2016

(Unterschrift Verfasser) (Unterschrift Betreuung)

Erklärung zur Verfassung der Arbeit

Harald Demel
Trazerberggasse 33/4/10, 1130 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasser)

Abstract

Batch queuing systems are frequently used for calculations consisting of a large number of similar, independent tasks. The total runtime can be reduced by parallel execution of tasks within the limits of the resources available. Runtime minimization is especially important when working towards deadlines, but it's not economic to increase the amount of resources for load peaks.

In this work, a batch queuing system is extended by autonomous scaling capabilities by developing interfaces for grid-computing and a cloud-based system. The requirements for security and monitoring are analyzed and the performance overhead is examined and compared to related works. Due to the hourly billing model of most cloud providers and the boot time of the nodes executing all tasks in parallel is not cost efficient for task lengths significantly below one hour. An algorithm is needed which minimizes the time until job completion while maintaining minimum cost-overhead. For an ideal solution, a-priori knowledge about the runtime of each task is required. In practice, however, task runtime cannot be known beforehand. Several heuristics to handle this problem are developed and compared. The autonomous scaling capabilities can also be used in a hybrid setup where a grid computing system is extended by scaling out to the cloud when too little resources are available on the grid.

For applications in the domain of nano-TCAD runtime and cost are compared for manually started nodes, grid-computing, a cloud-based and a hybrid system. These benchmarks show a significant reduction of the total runtime with the cloud-based system even when using the heuristic to minimize cost. The hybrid system delivers results faster than pure cloud or pure grid systems and costs less than the pure cloud system. These benefits only play out with small input files though, otherwise the file transmission time outweighs the benefits.

This work demonstrates that an autonomously scaling batch queuing system can deliver results faster by short term use of additional resources and maintain minimal cost at the same time. The presented hybrid system can reduce total runtime and cost compared to a pure cloud system.

Kurzfassung

In Batch-Queuing-Systemen werden häufig Berechnungen durchgeführt, die aus einer großen Anzahl von ähnlichen, voneinander unabhängigen Teilaufgaben bestehen. Durch parallele Abarbeitung kann die Berechnungszeit reduziert werden, allerdings nur im Rahmen der zur Verfügung stehenden Hardware. Die Reduktion der Berechnungszeit ist besonders relevant, wenn auf Terminvorgaben hingearbeitet wird. Eine permanente Erhöhung der Rechenkapazität für Lastspitzen ist jedoch nicht effizient.

Im Zuge dieser Arbeit wird ein Batch-Queuing-System um die Fähigkeit zur autonomen Skalierung erweitert indem jeweils eine Anbindung an ein Grid-Computing-System und eine IaaS-Cloud entwickelt wird. Die dadurch entstehenden Anforderungen an Security und Monitoring werden untersucht und der entstehende Performance-Overhead wird analysiert und mit anderen Arbeiten verglichen. Durch das stundenweise Abrechnungsmodell von IaaS-Clouds und die Hochlaufzeit der Rechenknoten ist es nicht kosteneffizient alle Teilaufgaben einer Berechnung parallel zu berechnen, wenn deren Laufzeit deutlich unter einer Stunde liegt. Ein Algorithmus welcher die Berechnungszeit minimiert und dabei die Kosten minimal hält wird benötigt. Für eine optimale Lösung sind Informationen über die Laufzeit der Teilaufgaben erforderlich, diese sind aber vorab nicht bekannt. Daher werden Heuristiken zur Lösung dieses Problems entwickelt und verglichen. Die Fähigkeit zur autonomen Skalierung kann außerdem in einem Hybridsystem eingesetzt werden. Dabei wird das Grid-Computing-System um die Fähigkeit erweitert bei unzureichenden Ressourcen in die IaaS-Cloud zu skalieren.

Anhand von Berechnungen aus dem Einsatzgebiet Nano-TCAD werden Berechnungsdauer und Kosten bei Verwendung von manuell gestarteten Rechenknoten, Grid-Computing-System, IaaS-Cloud und Hybridsystem gegenübergestellt. Dabei zeigt sich, dass die Berechnungszeit durch die Berechnung in der IaaS-Cloud, trotz Verwendung der Heuristik zur Kostenminimierung, reduziert werden kann. Ein Hybridsystem kann schneller Ergebnisse liefern als die Nutzung von Grid-Computing-System oder Cloud-Computing alleine und verursacht dabei geringere Kosten als die ausschließliche Nutzung von Cloud-Ressourcen. Diese Vorteile kommen allerdings nur bei einer geringen Größe der Eingabedaten zu Tragen, da sonst die für Datenübertragung benötigte Zeit die Vorteile überwiegt.

Es wird gezeigt, dass ein autonom skalierendes Berechnungsnetzwerk durch den kurzfristigen Zugriff auf mehr Ressourcen schneller Ergebnisse liefern kann als ein statisches und dabei die Kosten minimal gehalten werden können. Das entwickelte Hybridsystem kann sowohl Berechnungszeit als auch Kosten gegenüber Cloud-Computing verringern.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

KAPITEL 1

Einleitung

In Batch-Queuing-Systemen werden Einzelaufgaben (Jobs) in einer Warteschlange gesammelt und auf der zur Verfügung stehenden Hardware ausgeführt. In vielen Fällen bestehen Berechnungsaufgaben aus einer großen Anzahl von ähnlichen, voneinander unabhängigen Einzelaufgaben, deren Teilergebnisse dann zu einem Gesamtergebnis zusammengefasst werden. Die unabhängigen Einzelaufgaben können von einem Batch-Queuing-Systemen parallel abgearbeitet werden. Die Anzahl dieser Einzelaufgaben kann für große Berechnungsaufgaben aber schnell die Kapazitäten der zur Verfügung stehenden Hardware überschreiten. Dadurch werden die unabhängigen Einzelaufgaben nicht mehr alle parallel abgearbeitet und die Zeit bis zum Abschluss der Berechnung aller Einzelaufgaben verlängert sich.

Die Menge der zur Verfügung stehenden Hardware permanent zu erhöhen würde in diesen Fällen helfen. Das ist jedoch nicht effizient, wenn es sich um eine Lastspitze handelt. Eine geeignetere Lösung ist in diesem Fall die autonome Skalierung der Hardwareressourcen. Dies wird aber von den gängigen Batch-Queuing-Systemen nicht unterstützt [1] [2].

Im Zuge dieser Arbeit soll ein bestehendes Batch-Queuing-System [3] mit fester Menge an zur Verfügung stehender Hardware so erweitert werden, dass eine autonome Skalierung der Hardwareressourcen erfolgt. Dies geschieht zum einen durch das Starten neuer Rechenknoten in einem anderen Batch-Queuing-System. Dadurch erfolgt aus Sicht des ersten Systems eine Skalierung, obwohl dem Gesamtsystem keine weitere Hardware hinzugefügt wird. Der zweite in dieser Arbeit untersuchte Ansatz ist das Starten neuer Rechenknoten in “Infrastructure as a Service (IaaS) Clouds”. Das Ergebnis dieser Erweiterungen ist ein Batch-Queuing-System das Rechenknoten aus mehreren Rechenknotengruppen, von denen einige skalierbar sein können, transparent zu einem autonom skalierenden System zusammenfasst.

Die Bedeutung der Berechnungszeit ist hoch, wenn auf eine Deadline hingearbeitet wird, daher ist die Möglichkeit der Skalierung hier besonders relevant [4]. Auch wenn die Berechnungsaufgabe Teil eines iterativen Arbeitsprozesses ist, kann sich die Bedeutung der Ausführungszeit erhöhen. Ein Beispiel eines iterativen Arbeitsprozesses, in dem Berechnungsaufgaben

in Form von aufwendigen Simulationen vorkommen, ist die Entwicklung neuartiger Halbleiterbauelementstrukturen. Im Arbeitsprozess wechseln sich die Arbeitsschritte *Simulieren des Bauelements* und *Anpassen von dessen Parametern anhand der Ergebnisse der Simulation* ab. Die Entwicklerin oder der Entwickler muss also auf das Simulationsergebnis warten, um weiterarbeiten zu können. Die Wartezeit beeinflusst unter Umständen nicht nur die Geschwindigkeit, mit der die Entwicklung voranschreitet, sondern auch die Arbeitsweise des Entwickelns, da nicht mit Simulationen gespart werden muss. So ist es bei kürzeren Simulationszeiten eventuell möglich darauf zu verzichten, mehrere unterschiedliche Parameter in einem Anpassungsschritt zu verändern und stattdessen nach jeder Parameteränderung eine Simulation durchzuführen. Dadurch ist der Einfluss der einzelnen Parameter besser erkennbar, wodurch unter Umständen ein besseres Entwicklungsergebnis erzielt werden kann.

Aufbau der Arbeit:

Kapitel 2: Evaluierung und Definition der Anforderungen für die Erweiterungen zu einem autonom skalierenden System

Kapitel 3: Architektur und Umsetzung der Anbindungen zu IaaS-Cloud und Grid-Computing-System

Kapitel 4: Vergleich der Anbindungen miteinander, mit vorab manuell gestarteten Rechenknoten und mit einem Hybridsystem, das beide Anbindung verwendet

Kapitel 5: Qualitativer Vergleich der Erweiterungen zu einem autonom skalierenden System mit verwandten Arbeiten

Kapitel 6: Weiterführende Forschung zur Leistungssteigerung des Berechnungsnetzwerks

Kapitel 7: Zusammenfassung und Fazit

KAPITEL 2

Anforderungen an autonom skalierende Systeme

Beim Betrieb von Webseiten mit vielen Seitenaufrufen wird seit Jahren intensiv auf autonom skalierende Systeme gesetzt. Da bei Webseiten eine sofortige Bearbeitung der Anfrage erforderlich ist, muss hier zu jedem Zeitpunkt genügend Rechenkraft zur Verfügung stehen. Aufgrund der großen Lastschwankungen vieler Webseiten, zum Teil auf ein vielfaches der typischen Auslastung innerhalb von wenigen Minuten [5], wäre es aber unwirtschaftlich die benötigte Hardware anhand der Lastspitzen zu bestimmen. Ein großer Teil der Ressourcen würden dann nur zu den Spitzenzeiten genutzt. Unter anderem aus diesem Grund bietet sich dieses Anwendungsfeld besonders für autonome Skalierung in einer IaaS-Cloud oder eine Hybridlösung an.

Im Bereich von Batch-Queuing-Systemen sind die Anforderungen an die Menge der zur Verfügung stehenden Hardware nicht von so kritischer Bedeutung wie beim Betrieb von Webseiten, da das System nicht zusammenbricht wenn mehr Anfragen vorhanden sind als abgearbeitet werden können. Dennoch wollen die Benutzerinnen und Benutzer des Systems möglichst schnell Berechnungsergebnisse erhalten und dies kann auch wirtschaftlich große Unterschiede machen (siehe Kapitel 1 *Einleitung*). Da es auch bei Batch-Queuing-Systemen zu sehr hohen Lastschwankungen kommen kann, bieten sich auch hier Lösungen, die ganz oder teilweise IaaS-Clouds verwenden, an.

In IaaS-Clouds kostet es gleich viel 1000 Server für eine Stunde oder einen Server für 1000 Stunden zu mieten [6]. Das theoretisch optimale Ergebnis wäre also immer soviel Hardware anzumieten, dass alle unabhängigen Einzelaufgaben parallel ablaufen. In der Praxis müssen allerdings die Abrechnungsmodelle der Cloud-Provider und Wartezeiten beim Anlegen neuer Instanzen berücksichtigt werden [7]. So rechnen die Cloud-Provider in der Regel Stundenweise ab. Das heißt zu beginn jeder neuen Stunde Laufzeit entstehen zusätzliche Kosten. Handelt es sich also um 1000 kurze Jobs kostetet es also keineswegs gleich viel wenn für diese 1000 Rechenserver angefordert werden oder ob sie auf einer Maschine berechnet werden. Die Fragestellung wie viele Rechenserver optimal sind kann vorab nicht beantwortet werden, da die Ausführungszeit eines Jobs zwar eventuell vorab abgeschätzt, jedoch nicht mit Sicherheit bestimmt

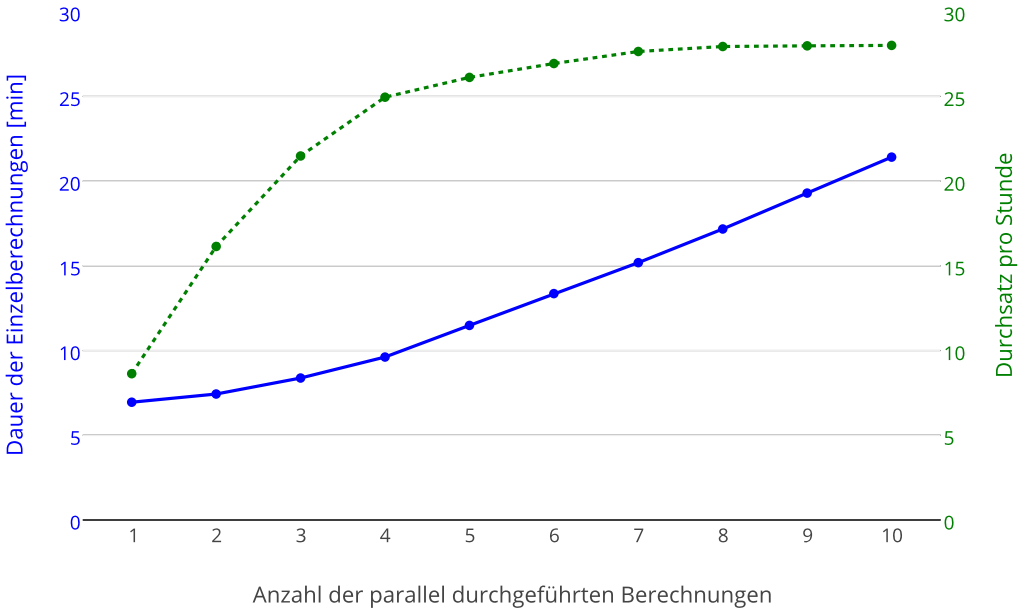

ABBILDUNG 2.1: ***Under-provisioning-Analyse:*** *Under-provisioning-Analyse: Auf einer Maschine mit einer CPU mit vier Kernen und acht Ausführungs-Threads (Intel Core i7-4790K) werden ein bis zehn identische Berechnungen parallel ausgeführt. Im Bereich von ein bis vier Berechnungen steigt der Durchsatz an Berechnungen pro Stunde (gestrichelte Linie) steil an, da jede Berechnung nur einen Kern benötigt. Steigert man die Berechnungen auf fünf bis acht, erhöht sich der Durchsatz nur noch leicht. Dies ist dadurch zu erklären, dass jeder Kern zwei Ausführungs-Threads besitzt und so im Fall von Wartezeiten bei Arbeitsspeicher- oder Festplattenzugriffen mit geringem Overhead auf den anderen Ausführungs-Thread umschalten kann. Wird die Anzahl der Berechnungen noch weiter erhöht, verbessert sich der Durchsatz kaum noch. Die Dauer der Einzelberechnungen (durchgezogenen Linie) erhöht sich im Bereich von ein bis vier parallel durchgeführten Berechnungen nur leicht, da für jede Berechnung ein CPU-Kern zur Verfügung steht. Werden mehr als vier Berechnungen parallel durchgeführt erhöht sich die Berechnungsdauer annähernd linear.*

werden kann. Außerdem ist nicht absehbar, wie viele neue Jobs während der Abarbeitung hinzukommen, die die angeforderte Hardware weiternutzen. Ein zweiter Faktor sind Wartezeiten beim Anlegen neuer Instanzen. Es ist also möglich, dass die angeforderte Hardware nicht mehr benötigt wird, wenn sie zur Verfügung steht. Außerdem muss die Zeit, welche von der Recheninstanz zum Hochfahren benötigt wird, auch bezahlt werden. Es ist also unter Umständen günstiger, die Hardwaremenge nicht unmittelbar an die Lastschwankungen anzupassen.

Eine weiterer Parameter um die Kosten für die Berechnung zu optimieren, ist die Anzahl der Aufgaben die von einem Rechenserver auf einmal bearbeitet werden. Wird nur eine Aufgabe bearbeitet, so kann die Skalierung sehr feingranular stattfinden und gegenseitige Beeinflussung von Aufgaben ist ausgeschlossen. Werden beispielsweise auf einer vier mal so leistungsfähigen Maschine bis zu vier Aufgaben parallel bearbeitet, so hat dies unter anderem die Vorteile, dass der Overhead des Betriebssystems für alle vier Aufgaben in Summe nur einmal anfällt. Außerdem ist es möglich, dass verschiedene gleichzeitige Berechnungen durch unterschiedliche Hardwareparameter limitiert werden. Beispiele dafür sind Arbeitsspeichernutzung, CPU-Auslastung und Festplattenzugriffe. Wenn also eine Berechnung mit hohem Bedarf an Arbeitsspeicher, zwei mit viel CPU-Bedarf und eine mit einer großen Anzahl an Festplattenzugriffen auf dem gleichen Rechenserver parallel abgearbeitet werden ist dies in der Regel schneller als auf vier getrennten Maschinen.

Eine weitere Möglichkeit um sicherzustellen, dass die Hardware optimal ausgenutzt wird, ist ein gewisses Maß an Under-provisioning. Werden auf einer Maschine mit vier CPUs nicht nur vier sondern fünf, sechs oder sieben Aufgaben parallel bearbeitet, so wird sichergestellt, dass die CPUs immer voll ausgelastet sind, auch wenn ein Prozess auf Arbeitsspeicher- oder Festplattenzugriffe wartet (siehe Abbildung 2.1).

Durch dieses Under-provisioning kann die Effizienz der eingesetzten Hardware zwar erhöht werden, jedoch verlängert sich die Ausführungszeit der einzelnen Jobs. Dies widerspricht dem ursprünglichen Ziel, die Wartezeit für die Benutzerin oder den Benutzer möglichst gering zu halten. Daher sollte nur so wenig Under-provisioning wie nötig benutzt werden, um die Effizienz zu verbessern. In dem in Abbildung 2.1 gezeigten Beispiel wäre also Under-provisioning mit fünf oder sechs parallel bearbeiteten Aufgaben bei hoher Priorität von kurzen Wartezeiten, oder sieben parallel bearbeiteten Aufgaben bei hoher Priorität von hohem Durchsatz zu wählen. Das Minimum an möglichem Under-provisioning ist offensichtlich, einer Maschine um eine Aufgabe mehr zu geben als diese CPUs hat. Bei einer geringen Anzahl an CPUs ist das minimal mögliche Under-provisioning relativ gesehen also größer als bei Maschinen mit höherer Anzahl an CPUs. Dies ist ein weiteres Argument für stärkere Maschinen mit mehreren gleichzeitigen Berechnungen.

2.1 Security

Da es möglich und in vielen Anwendungsfällen notwendig ist, dass die Komponenten des Batch-Queuing-Systems über das Internet kommunizieren, ist es essentiell, mögliche Bedrohungsszenarien zu analysieren. Dazu muss zuerst geklärt werden, welche Informationen und Komponenten schützenswert sind. Je nach Anwendung können sowohl die Ausgangsdaten der Berechnung

als auch die Ergebnisse Betriebsgeheimnisse sein. Außerdem müssen der zentrale Koordinationsserver und alle Berechnungsknoten geschützt werden, da die vertraulichen Daten dort verarbeitet werden.

2.1.1 Bedrohungsszenarien von Applikationen im Internet

Im Folgenden sollen zutreffende Bedrohungsszenarien von Applikationen im Internet aus [8] aufgelistet und Maßnahmen zur Absicherung gesucht werden.

2.1.1.1 Einschleusen von ausführbarem Code (Injection Flaws)

Das Batch-Queuing-System ist so aufgebaut, dass autorisierte BenutzerInnen beliebige Dateien zum Rechenknoten senden und dort beliebige Befehle ausführen können, das Einschleusen von ausführbarem Code ist also explizit gewollt. Wird ein zentraler Koordinationsserver von mehren BenutzerInnen verwendet so können diese untereinander Dateien einsehen und verändern, das kann innerhalb von Arbeitsgruppen akzeptabel sein. Alternativ kann pro BenutzerIn ein zentraler Koordinationsserver gestartet werden. Das Einschleusen von ausführbarem Code ist somit weiterhin möglich, jedoch ist der Wirkungsbereich durch das BenutzerInnenmanagementsystem des Betriebssystems eingeschränkt. Einschleusen durch AngreiferInnen wird dadurch verhindert, dass vor dem Starten von Berechnungen Authentifizierung erforderlich ist.

2.1.1.2 Ausführung von bösartig manipulierten Dateien (Malicious File Execution)

Siehe vorheriger Abschnitt.

2.1.1.3 Informationsleck durch ungeeignete Fehlerbehandlung (Information Leakage by Improper Error Handling)

Wie bereits beschrieben ist das Batch-Queuing-System nicht gegen autorisierte BenutzerInnen geschützt. Der Fokus liegt also am Authentifizierungsvorgang.

2.1.1.4 Unsichere Kommunikation (Insecure Communication)

Die Kommunikation aller Komponenten des Batch-Queuing-Systems miteinander muss verschlüsselt sein.

2.1.1.5 Defekte Authentifizierung und Sitzungsmanagement (Broken Authentication and Session Management)

Selbst wenn die Kommunikation verschlüsselt ist, kann sich eine Angreiferin oder ein Angreifer durch einen aktiven Angriff in die Kommunikation einschleusen, falls keine Authentifizierung der Kommunikationspartner stattfindet (Man-in-the-Middle-Angriff). Dies kann verhindert werden, indem die Authentizität des Servers von allen Komponenten sichergestellt wird, die sich zu diesem verbinden. Umgesetzt werden kann dies, indem diese Komponenten den öffentlichen

Schlüssel des Servers kennen.

Da Clients nicht nur neue Berechnungen in Auftrag geben können, sondern sich auch die Ergebnisse zu abgeschlossenen Berechnungen abholen können, ist es notwendig den Zugriff mittels BenutzerInnenkonten zu verwalten. Aber nicht nur der unautorisierte Zugriff von Clients muss verhindert werden, sondern auch das Registrieren als Rechenknoten muss abgesichert werden. Wäre es möglich sich ohne Absicherung als Rechenknoten zu registrieren, dann könnte eine Angreiferin oder ein Angreifer an die Ausgangsdaten einer Berechnung gelangen. Daher muss auch das Registrieren von Rechenknoten mittels BenutzerInnenkonten oder einem ähnlichen Mechanismus abgesichert werden.

Die Authentifizierung der Rechenknoten wird dadurch erschwert, dass diese auch ohne BenutzerInneninteraktion starten können sollen. Bei Rechenknoten die vom Koordinationsserver gestartet werden, um die Rechenkapazität nach oben zu skalieren, kann die Authentifizierung durch die Verwendung eines Einmalpassworts gelöst werden. In einem Hybridsystem, bei dem ein Grundstock an Rechenkapazität ständig zur Verfügung stehen soll, melden sich die Rechenknoten jedoch selbstständig nach einem Neustart wieder an. Dabei ist Authentifizierung durch Einmalpasswörter nicht anwendbar.

Die Benutzung von BenutzerInnenname und Passwort würde es nötig machen diese im Klartext zu speichern. Dadurch wäre es schwierig die Kontrolle darüber zu behalten wer Zugang zu diesen Informationen hat. Außerdem führt ein solches System bei vielen Rechenknoten dazu, für alle Knoten das selbe, einfache Passwort zu verwenden, wodurch es unmöglich ist, einzelne Knoten sicher aus dem System zu entfernen.

Die Anforderungen an das Authentifizierungssystem sind also:

- Der Administrator muss nur ein Passwort kennen, egal wie viele Rechenknoten das System hat.
- Es ist möglich einem einzelnen Knoten die Berechtigungen zu entziehen, ohne auf einem anderen Knoten Änderungen vorzunehmen.
- Werden Anmeldeinformationen von der Festplatte eines Knoten gestohlen, hat dies keine Auswirkungen auf die anderen Knoten.
- Werden gestohlene Anmeldeinformationen von einer Angreiferin oder einem Angreifer verwendet, wird dies vom System erkannt.

Ein mögliches Authentifizierungssystem, das diese Anforderungen erfüllt, kann mit Hilfe von zwei Zufallszahlen umgesetzt werden [9]. Bei der Einrichtung der Installation durch den Administrator gibt dieser einmalig seinen BenutzerInnenname und sein Passwort ein. Sind diese Anmeldedaten gültig, so wird am Koordinationsserver eine Seriennummer und ein Token generiert, gespeichert, an den Rechenknoten gesendet und dort auf die Festplatte geschrieben. Bei jeder weiteren Anmeldung sendet der Rechenknoten nun Seriennummer und Token. Sind diese korrekt, generiert der Koordinationsserver ein neues Token und sendet es zurück an den Rechenknoten, die Seriennummer bleibt jedoch gleich.

Wird nun die Anmeldeinformationen von der Festplatte des Rechenknotens gestohlen und von der Angreiferin oder dem Angreifer verwendet, so bekommt die Angreiferin oder der Angreifer ein neues Token und das am Rechenknoten gespeicherte Token wird ungültig. Meldet sich der Rechenknoten nun erneut an, stellt der Koordinationsserver fest, dass der Rechenknoten zwar eine gültige Seriennummer jedoch ein ungültiges Token hat. Da dies nur dadurch erklärt werden kann, dass die Anmeldeinformationen gestohlen wurden, sperrt der Koordinationsserver sofort jeden weiteren Zugriff dieser Seriennummer und alarmiert den Administrator.

Da jeder Rechenknoten eine eigene Seriennummer besitzt, hat der Kontrollverlust über die Anmeldeinformationen keine Auswirkungen auf die anderen Knoten. Dadurch können auch einem einzelnen Knoten die Berechtigungen entzogen werden indem der jeweilige Seriennummerneintrag gelöscht wird. Erleichtert wird dies, wenn bei der Speicherung der Seriennummer der Name des Rechenknotens mitgespeichert wird.

2.1.2 Bedrohungsszenarien von Applikationen in der Cloud

Im Folgenden sollen Bedrohungsszenarien von Applikationen in der Cloud aus [8] aufgelistet, auf ihr Zutreffen untersucht und gegebenenfalls Maßnahmen zur Absicherung gesucht werden.

2.1.2.1 Erschöpfung der Ressourcen (Resource Exhaustion)

Bei einer Erschöpfung der Cloud-Ressourcen werden BenutzerInnen-Anfragen nach zusätzlichen Ressourcen nicht bearbeitet. Dadurch kann die Applikation nicht wie geplant skalieren. Bei einem Batch-Queuing-System führt das zu einer Verzögerung der Fertigstellung von Berechnungen aber nicht zum Kollaps des Systems.

2.1.2.2 Fehler in der Mandantenfähigkeit des Cloud-Anbieters (Isolation Failure)

Bei Fehler in der Mandantenfähigkeit können durch die gemeinsame Nutzung von Rechenkapazität, Speicher und Netzwerk Datenlecks entstehen. Eine Möglichkeit dieses Problem zu verringern ist das Mieten von ganzen Maschinen. Ist dies nicht wirtschaftlich da dadurch die Schrittweite bei der Anpassung der Hardwaremenge zu groß wäre, kann das Risiko minimiert werden indem nicht mehr benötigte Daten so früh wie möglich gelöscht werden.

2.1.2.3 Bösartige(r) MitarbeiterIn des Cloud-Anbieters (Cloud Provider Malicious Insider)

Bei der Nutzung von virtualisierter Hardware eines IaaS-Cloud-Providers ist es für den Provider jederzeit möglich einen Snapshot der Maschine zu machen und somit den Inhalt der Festplatte und des Arbeitsspeichers einzusehen [10]. Will man die Betriebsgeheimnisse dem IaaS-Cloud-Provider nicht anvertrauen, so dürfen diese also niemals auf die Festplatte oder in den Arbeitsspeicher gelangen. Dies ist mit homomorpher Verschlüsselung [11] zu bewerkstelligen. Dabei werden die Berechnungen auf den verschlüsselten Daten durchgeführt. Das Ergebnis kann somit in immer noch verschlüsselter Form zur Benutzerin oder zum Benutzer zurückgesendet und erst dort entschlüsselt werden.

Bei partiell homomorpher Verschlüsselung, welche mit annehmbarem Aufwand möglich ist, ist allerdings nur ein Subset an Rechenoperationen möglich [12]. Wegen dieser Einschränkung ist sie nur für sehr stark abgegrenzte Aufgaben verwendbar, nicht in einem System für allgemeine Berechnungsaufgaben.

Voll-homomorphe Verschlüsselungssysteme erlauben eine Vielzahl an Rechenoperationen, erhöhen den Rechenaufwand im Vergleich zur unverschlüsselten Berechnung aber um ein Vielfaches. Da Batch-Queuing-Systeme aber gerade für Berechnungen genutzt werden die ohnehin schon sehr aufwendig sind und daher viel Zeit zur Berechnung benötigen, ist eine weitere Erhöhung des Aufwands in diesem Ausmaß nicht akzeptabel.

Zur Zeit gibt es also keine brauchbare technische Lösung für dieses Problem. Es bleiben also nur die Möglichkeiten auf virtualisierte Hardware eines IaaS-Cloud-Providers zu verzichten oder dem Provider zu Vertrauen beziehungsweise sich vertraglich abzusichern. Für den Fall, dass trotzdem eine IaaS-Cloud genutzt werden soll, kann zwar keine Sicherheit garantiert werden, das Risiko eines Datenverlusts kann jedoch minimiert werden. Dies kann beispielsweise durch Datensparsamkeit erreicht werden. Ist eine Berechnung auf einem Rechenknoten abgeschlossen und sind die Ergebnisse übermittelt, so sollten unverzüglich alle Daten der Berechnung, die ja nun nicht mehr am Rechenknoten gebraucht werden, gelöscht werden.

Die selben Risiken die für virtualisierte Hardware beschrieben wurden treffen auch auf physische Hardware zu, zu der außenstehenden Personen physischen Zugang haben. Sollen alle möglichen Maßnahmen getroffen werden, um die Berechnungsdaten zu schützen, muss sich also sämtliche Hardware innerhalb des Unternehmens befinden und der Zugang abgesichert sein.

2.1.2.4 Kompromittierung der Verwaltungsschnittstelle (Management Interface Compromise)

Grundvoraussetzung für die Sicherheit der Verwaltungsschnittstelle ist die Authentifizierung des Cloud-Providers beim Verbindungsaufbau und die Verschlüsselung der Verbindung. Eine weitere potentielle Schwachstelle stellen Zugangsdaten dar, die in Konfigurationsdateien eingetragen sind. Der Zugriff auf diese Dateien muss entsprechend eingeschränkt werden.

2.1.2.5 Abfangen von Datenübertragungen (Intercepting data in Transit)

Sowohl bei der Kommunikation mit dem Cloud-Provider als auch der Komponenten untereinander muss ausnahmslos Authentifizierung und Verschlüsselung eingesetzt werden.

2.1.2.6 Datenleck bei der Übertragung in die/aus der Cloud (Data Leakage on Up/Down)

Auch beim Zugriff auf die in der Cloud ausgeführten Komponenten von Außerhalb muss beidseitige Authentifizierung und Verschlüsselung eingesetzt werden.

2.1.2.7 Unsicheres oder ineffizientes Löschen von Daten (Insecure or Ineffective Deletion of data)

Um das Risiko eines Datenlecks zu minimieren sollten nicht mehr benötigte Daten so früh wie möglich gelöscht werden. Eine Garantie, dass das Löschen auf der virtuellen Festplatte auch ein unmittelbares physisches Löschen auf allen Speichermedien zufolge hat gibt es allerdings nicht.

2.1.2.8 Distributed Denial of Service

Durch die öffentliche Erreichbarkeit der eingesetzten Maschinen können diese mit einem Distributed Denial of Service angegriffen werden. Das kann nicht verhindert werden, die Widerstandsfähigkeit gegen "Denial of ServiceAngriffe kann jedoch erhöhrt werden indem der Datenstrom, beispielsweise durch eine Firewall, gefiltert wird.

2.1.2.9 Ökonomischer Denial of Service (Economic Denial of Service)

Nachdem ein zentraler Koordinationsserver nur von einer geschlossenen Gruppe an authentifizierten BenutzerInnen genutzt wird kann die Menge der genutzten Ressourcen nicht von Außenstehenden beeinflusst werden, vorausgesetzt die Benutzerkennungen bleiben vertraulich.

2.1.2.10 Verlust des Verschlüsselungsschlüssels (Loss of Encryption Keys)

Sowohl der Schlüssel zur Authentifizierung gegenüber dem Cloud-Provider als auch der private Schlüssel des zentralen Koordinationsservers müssen geheim gehalten werden. Dazu sollten die Schlüssel so abgelegt werden, dass von innerhalb des Batch-Queuing-Systems nicht zugreifbar sind.

2.1.2.11 Untersuchung auf Schwachstellen (Undertaking malicious probes or scans)

Durch die öffentliche Erreichbarkeit der eingesetzten Maschinen können diese bösartig auf Schwachstellen untersucht werden und somit ein Angriff vorbereitet werden. Um dieses Risiko einzudämmen sollte der Zugriff von außen mittels einer Firewall auf die unbedingt notwendigen Dienste beschränkt werden.

2.1.2.12 Schwachstelle der Cloud-Virtualisierungsschicht (Compromise service engine)

Bei Schwachstellen der Cloud-Virtualisierungsschicht können andere Cloud-BenutzerInnen unter Umständen auf die eigenen Ressourcen oder Daten zugreifen. Da die Cloud-Virtualisierungsschicht nicht unter der eigenen Kontrolle liegt kann darauf wenig Einfluss genommen werden. Eine Möglichkeit dieses Problem zu vermeiden ist das Mieten von ganzen Maschinen. Ist dies nicht wirtschaftlich da dadurch die Schrittweite bei der Anpassung der Hardwaremenge zu groß wäre, kann das Risiko minimiert werden indem nicht mehr benötigte Daten so früh wie möglich gelöscht werden.

2.1.2.13 Konflikte zwischen Maßnahmen zur Härtung durch den Kunden und der Cloud-Umgebung (Conflicts between customer hardening procedures and cloud environment)

Cloud-Kunden müssen sich darüber bewusst sein, dass sie für die Sicherheit der eingesetzten Systeme verantwortlich sind und nicht der Cloud-Provider die Sicherheit garantiert. Auch die vom Cloud-Provider angebotenen Maschinenabbilder sind nicht unbedingt von Haus aus sicher konfiguriert [13]. Deshalb ist es hilfreich wenn der Cloud-Provider klar definiert welche Schritte zur Härtung der Systeme schon abgedeckt sind und wo noch Handlungsbedarf durch den Kunden besteht.

Die Härtung der Systeme wird durch die gemeinsame Verwendung von Ressourcen mit anderen Nutzern erschwert. Insbesondere die Netzwerkinfrastruktur kann in einer Cloud-Umgebung nicht so strikten Maßnahmen zur Härtung unterzogen werden wie in einem lokalen Datenzentrum da unterschiedliche Systemkomponenten über das Netzwerk kommunizieren. Gleichzeitig ist die Netzwerkinfrastruktur gegenüber anderen Cloud-Kunden exponiert. Daher muss hier besonderes Augenmerk auf feingranulare Kontrolle und Härtung gelegt werden.

2.2 Monitoring und Kundenabrechnung

Aufgrund der Verteilung über mehrere Rechenknoten kann der Berechnungsablauf ein und derselben Berechnungsaufgabe bei mehreren Durchläufen unterschiedlich sein. Außerdem ist der Zustand des Gesamtsystems nicht immer auf den ersten Blick erkennbar. Umso wichtiger ist es, Monitoring in die Software zu integrieren. Die Ausgaben des Monitorings sollen dazu verwendet werden, Funktionsstörungen zu beheben, die Performance zu verbessern bzw. Benchmarks zu generieren.

Weiters müssen Daten für die Kundenabrechnung bzw. die Limitierung der BenutzerInnenressourcen erfasst werden. Dazu ist es notwendig die Berechnungsdauer jeder Berechnung aufzuzeichnen. Die Umsetzung von Monitoring und Billing kann aufgrund der ähnlichen Anforderungen eventuell in einem gemeinsamen Modul erfolgen.

KAPITEL 3

Methoden für verteiltes Rechnen

Um Berechnungen verteilt durchführen zu können wird eine große Anzahl an Maschinen benötigt. Diese können unter anderem in einer IaaS-Cloud oder einem Grid-Computing-System organisiert werden. Während in einer IaaS-Cloud virtuelle Maschinen auf Stundenbasis gemietet werden, werden in einem Grid-Computing-System Einzelberechnungen in einem von mehreren Berechnungsslots einer Maschine durchgeführt. Ein bestehendes Batch-Queuing-System [3] soll um jeweils eine Anbindung für diese beide Methoden des verteilen Rechnens erweitert werden.

Um der Benutzerin oder dem Benutzer die Möglichkeit zu geben beim Starten einer Berechnung einen oder mehrere IaaS-Cloud-Anbieter oder Grid-Computing-Rechencluster wählen zu können, wurde das Konzept von Rechenknotengruppen eingeführt. Rechenknotengruppen werden dabei über eine Konfigurationsdatei definiert. Jede Rechenknotengruppe hat dabei einen Typ und eine Vielzahl von Parametern welche die Eigenschaften der Gruppe festlegen. Werden mehrere Rechenknotengruppen gleichzeitig ausgewählt, so müssen diese mit einem Kostenwert versehen werden. Einzelaufgaben, die auf ihre Ausführung warten, werden dann der Rechenknotengruppe mit dem niedrigsten Kostenwert zugeteilt. Diese kann die Einzelaufgabe dann bearbeiten oder an die nächst teurere Rechenknotengruppe weitergeben. In Kombination mit den Parametern der einzelnen Rechenknotengruppen, die darauf Einfluss nehmen, ob die Aufgaben angenommen oder weitergegeben werden, können auf diese Art vielfältige Verhaltensweisen mit einer unbeschränkten Anzahl an Rechenknotengruppen konfiguriert werden.

3.1 IaaS-Cloud-Anbindung

Im Zuge der Entwicklung der IaaS-Cloud-Anbindung wurde das bestehende Batch-Queuing-System (siehe Abschnitt 4.2 *Einsatz im Nano-TCAD*) so erweitert, dass je nach Bedarf zusätzliche Rechenknoten in einer IaaS-Cloud gestartet oder herunterfahren werden können (siehe Abbildung 3.1). Dafür interagiert die Anbindung mit der API von Amazon EC2 [14].

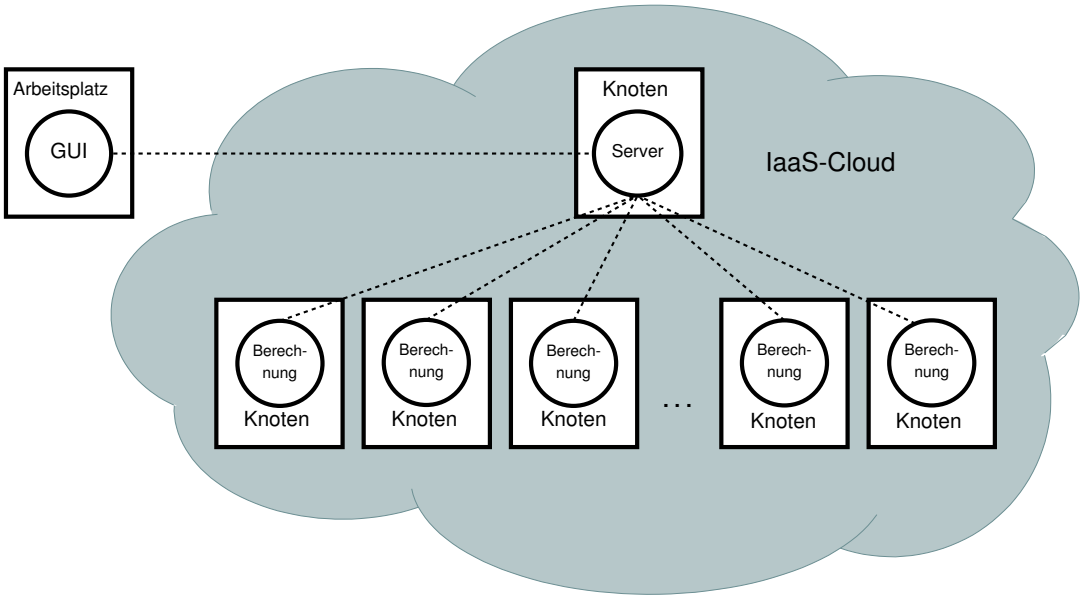

ABBILDUNG 3.1: ***Aufbau des Rechennetzwerks in einer IaaS-Cloud:*** *Die zentrale Koordinationsserversoftware kann innerhalb oder außerhalb der Cloud ausgeführt werden. Nachdem Berechnungsaufgaben von der grafischen BenutzerInnenoberfläche (GUI) übermittelt wurden, startet die IaaS-Cloud-Anbindung des zentrale Koordinationsservers via API Knoten in der IaaS-Cloud. Diese verbinden sich mit Hilfe der Berechnungsknotensoftware zum Koordinationsserver und arbeiten Teilberechnungen ab.*

3.1.1 Grundlegende Architektur

Wie bereits beschrieben soll die Menge der angemieteten IaaS-Cloud-Maschinen laufend dem Bedarf an Rechenknoten angepasst werden. Dabei müssen jedoch die Randbedingungen der stundenweisen Abrechnung und Hochlaufzeit der Rechenknoten berücksichtigt werden (siehe Kapitel 2 *Anforderungen an autonom skalierende Systeme*). Um diese Anforderungen zu erfüllen wird ein Algorithmus benötigt, der die Bearbeitungszeit der Berechnungsaufgabe minimiert während gleichzeitig die Kosten minimal gehalten werden. Für eine ideale Lösung muss die Laufzeit jeder Einzelaufgabe vorab bekannt sein. In der Praxis ist diese Information jedoch vorab nicht verfügbar. Der Algorithmus muss also zyklisch die Anzahl der benötigen Rechenknoten abschätzen und diese in der IaaS-Cloud anfordern, während die bereits angeforderten Knoten nach und nach hochfahren und der Rechenknotengruppe hinzugefügt werden.

Ein naiver Ansatz um die Menge zusätzlich zu startender Rechenknoten abzuschätzen, besteht darin, einen festen Prozentsatz der Anzahl wartender Teilaufgaben zu verwenden. Eine vereinfachte Form der Berechnungsformel, bei der angenommen wird, dass eine Maschine zu jedem Zeitpunkt nur eine Aufgabe bearbeiten kann ist $n_{\text{zusätzlich}} = \lceil n_{\text{wartend}} \times 10\% \rceil - n_{\text{hochfahrend}}$. Diese Vorgehensweise passt die Menge an Knoten nicht sprunghaft sondern schrittweise an den Bedarf an. Trotzdem liegen alle Berechnungsergebnisse innerhalb kurzer Zeit vor. Durch diesen Verzögerungseffekt werden bei Teilaufgaben mit sehr sehr kurzer Berechnungsdauer weniger Knoten gestartet als dies bei unverzögerter Anpassung der Fall wäre, der Overhead an Kosten

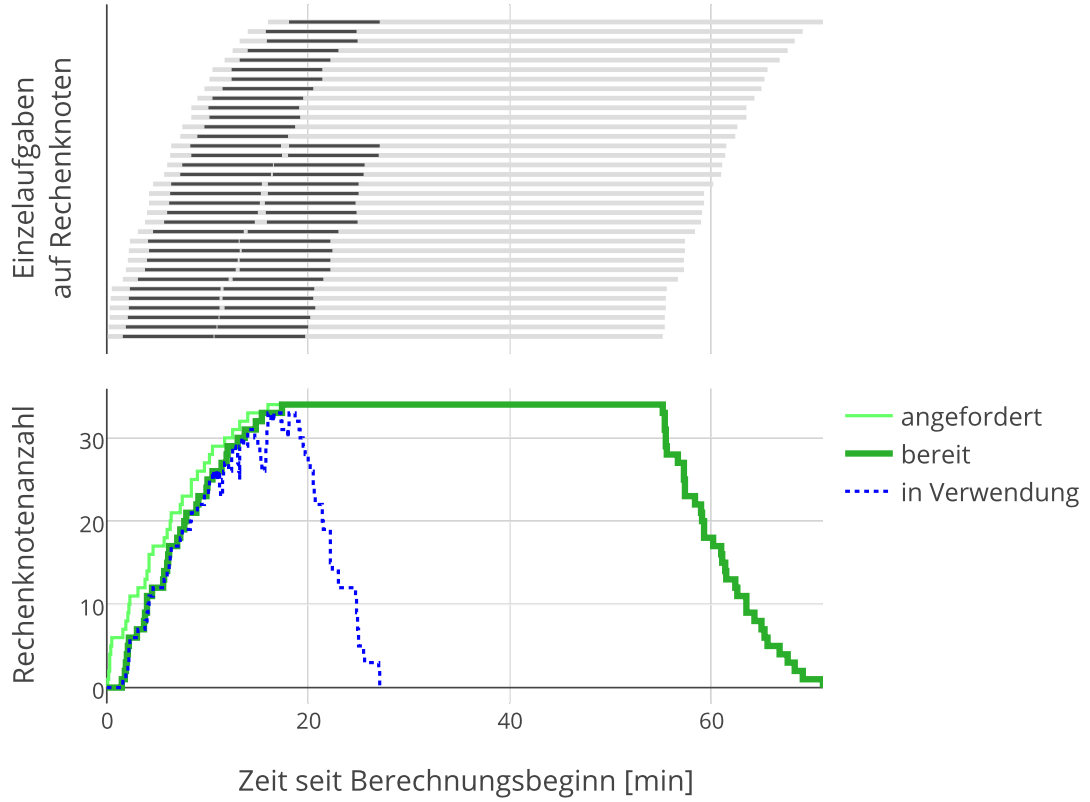

ABBILDUNG 3.2: ***Ablauf bei Verwendung des naiven Ansatzes mit kurzen Einzelaufgaben:*** *Visualisierung des Hochfahrens von Rechenknoten unter Verwendung des naiven Ansatzes bei 55 Einzelaufgaben mit je 9 Minuten Berechnungsdauer und einer simulierten Bootzeit von 90 Sekunden. Durch das verzögerte Hochfahren wird die Anzahl der gestarteten Rechenknoten von 55 auf 33 reduziert und damit enstehen Kosten für 33 Maschinenstunden. Die minimale benötigte Anzahl an Maschinenstunden beträgt allerdings lediglich zehn.*

kann allerdings immer noch ein vielfaches der optimalen Kosten betragen (siehe Abbildung 3.2). Den Prozentsatz weiter zu reduzieren, verkleinert den Overhead nicht signifikant und verlängert die Gesamtberechnungsdauer insbesondere bei vielen Knoten und langen Berechnungsaufgabe deutlich. Die starke Verlängerung der Gesamtberechnungsdauer kommt dadurch zustande, dass das Starten der letzten Rechenknoten stark verzögert wird (siehe Abbildung 3.3).

Eine verbesserter heuristischer Ansatz berechnet die erwartete Laufzeit t_{erwartet} für alle Teilaufgaben, basierend auf Informationen von bereits abgeschlossenen und noch laufenden vergleichbaren Berechnungsteilaufgaben. Die Menge der zusätzlich zu startenden Rechenknoten wird dann mittels der Formel

$$n_{\text{zusätzlich}} = \left\lceil \frac{\sum_{\text{Teilaufgabe}}(t_{\text{erwartet}} - t_{\text{ausgeführt}}) - \sum_{\text{Knoten}}(t_{\text{Abrechnungseinheit}} - t_{\text{Laufzeit}})}{50\,\text{min}} \right\rceil - n_{\text{hochfahrend}} \quad (3.1)$$

berechnet. Dabei ist $t_{\text{ausgeführt}}$ die Dauer, die eine Teilaufgabe bereits aufgeführt wird und $t_{\text{Abrechnungseinheit}} - t_{\text{Laufzeit}}$ ist die Zeit, die ein Rechenknoten noch laufen kann, bis er weitere Kosten verursacht. Diese Heuristik reduziert den Overhead an Kosten für kurze Teilaufgaben,

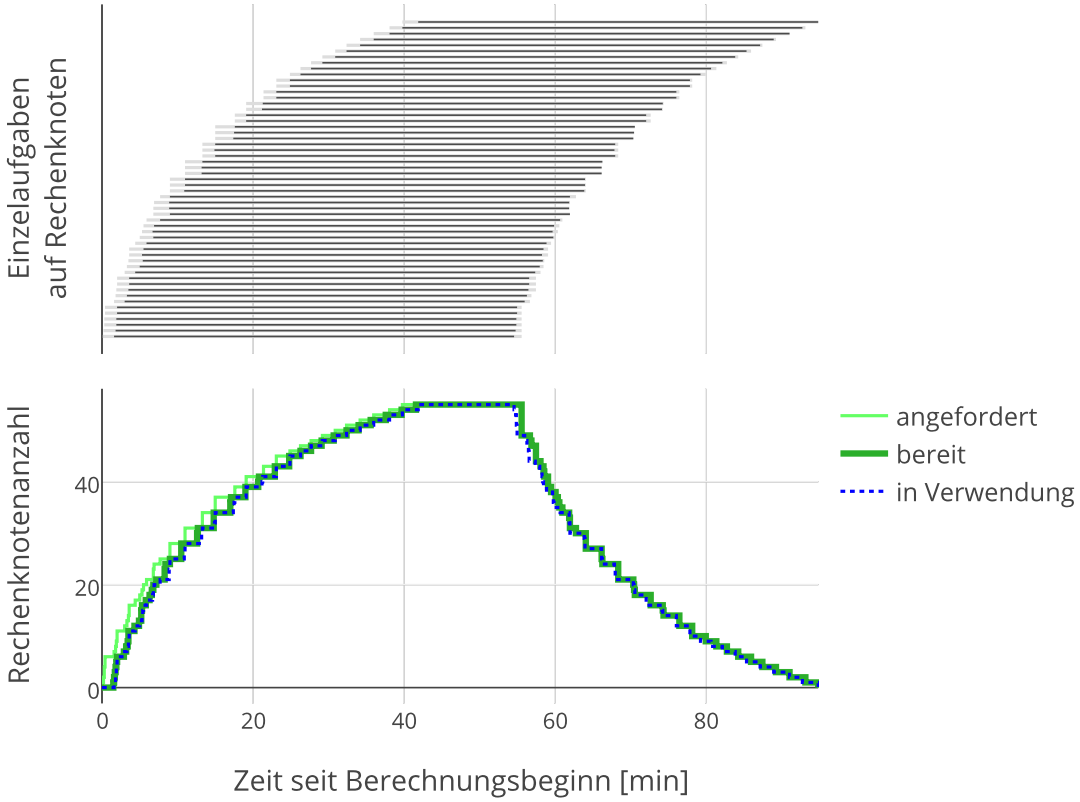

ABBILDUNG 3.3: ***Ablauf bei Verwendung des naiven Ansatzes mit langen Einzelaufgaben:*** *Visualisierung des Hochfahrens von Rechenknoten unter Verwendung des naiven Ansatzes bei 55 Einzelaufgaben mit je 53 Minuten Berechnungsdauer und einer simulierten Bootzeit von 90 Sekunden. Da die Berechnungsdauer beinahe so lang ist wie die Abrechnungseinheit soll für jede Maschine ein eigener Berechnungsknoten gestartet werden. Bei Verwendung des naiven Ansatz wird der letzte Berechnungsknoten erst nach über 39 Minuten gestartet und die Gesamtberechnungsdauer dadurch verlängert.*

erhöht die Gesamtberechnungsdauer jedoch über dafür nötige Maß hinaus, da Fragmentierung nicht berücksichtigt wird Abbildung 3.4.

Um dieses Fragmentierungsproblem zu vermeiden, kann der heuristische Ansatz um einen einfachen, schnellen Scheduling-Algorithmus erweitert werden. Dadurch wird die Gesamtberechnungsdauer verringert, da der Ressourcenbedarf früher erkannt wird und die entsprechenden Rechenknoten früher angefordert werden. In Abbildung 3.4 werden die Funktionsweisen der beiden heuristischen Ansätze gegenübergestellt und die Vorteile der Heuristik mit Scheduler gezeigt. Im Angeführten Beispiel wird eine Verringerung der Gesamtberechnungsdauer um 6.4% gezeigt. In Fällen mit einer höheren Anzahl an Einzelberechnung können aber weit höhere Unterschiede in der Gesamtberechnungsdauer auftreten, bis zu über 27% Verringerung (siehe Abbildung 3.5).

Eine Messreihe bei der die Berechnungsdauer von 55 Einzelaufgaben zwischen einer Minute und 53 Minuten variiert wurde (siehe Abbildung 3.5) zeigt, dass die beiden verbesserten heuristischen Ansätze für Berechnungen, deren Teilaufgaben eine Berechnungsdauer von weniger

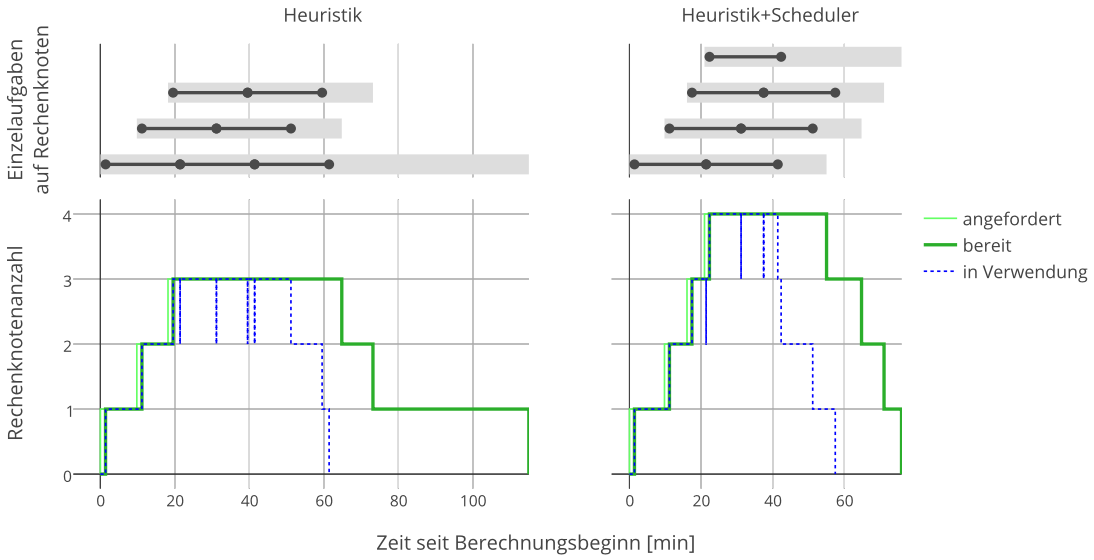

ABBILDUNG 3.4: ***Ablauf bei Verwendung der Heuristik mit und ohne Scheduling:*** *Gegenüberstellung des Berechnungsablaufs von sieben Einzelaufgaben mit je 20 Minuten Berechnungsdauer bei Verwendung der Heuristik zum Starten von Rechenknoten ohne (links) und mit (rechts) Scheduling. In beiden Fällen wird der erste Rechenknoten sofort, der zweite nach 9.8 Minuten gestartet. Der dritte Rechenknoten wird bei Verwendung der Heuristik ohne Scheduler (links) erst nach 18.2 Minuten gestartet, da die Summe der erwarteten verbleibenden Berechnungsdauern bis dahin in der Restlaufzeit der schon verfügbaren Rechenknoten untergebracht werden kann. Die Heuristik mit Scheduler (rechts) erkennt hingegen, dass die Anzahl der verbleibenden Einzelaufgaben ungerade ist und daher nicht gleichmäßig auf die Restlaufzeit der beiden vorhandenen Rechenknoten aufgeteilt werden kann. Daher wird der dritte Rechenknoten hier (rechts) schon nach 16.1 Minuten gestartet. Da nach 20.9 Minuten keine Aufteilung der erwarteten verbleibenden Berechnungsdauern auf die Restlaufzeit der drei Rechenknoten mehr möglich ist, wird schließlich der vierte Rechenknoten gestartet (rechts). Die Heuristik ohne Scheduler kann das nicht erkennen und startet keinen vierten Rechenknoten. Daher wird die Einzelberechnung, die rechts auf dem vierten Rechenknoten durchgeführt werden kann, links auf dem ersten Rechenknoten durchgeführt, sobald die anderen Berechnungen abgeschlossen sind. Das führt dazu, dass die Rechenknotenlaufzeit die Abrechnungseinheit von 60 Minuten überschreitet und Kosten für die neu angefangene Abrechnungseinheit entstehen. Es fallen also in beiden Fällen die gleichen Kosten von vier Abrechnungseinheiten an. Die Heuristik mit Scheduler liefert das Gesamtberechnungsergebnis schon nach 57.5 statt 61.4 Minuten. Das sind 6.4% Verringerung der Wartezeit auf das Ergebnis bei gleichen Kosten.*

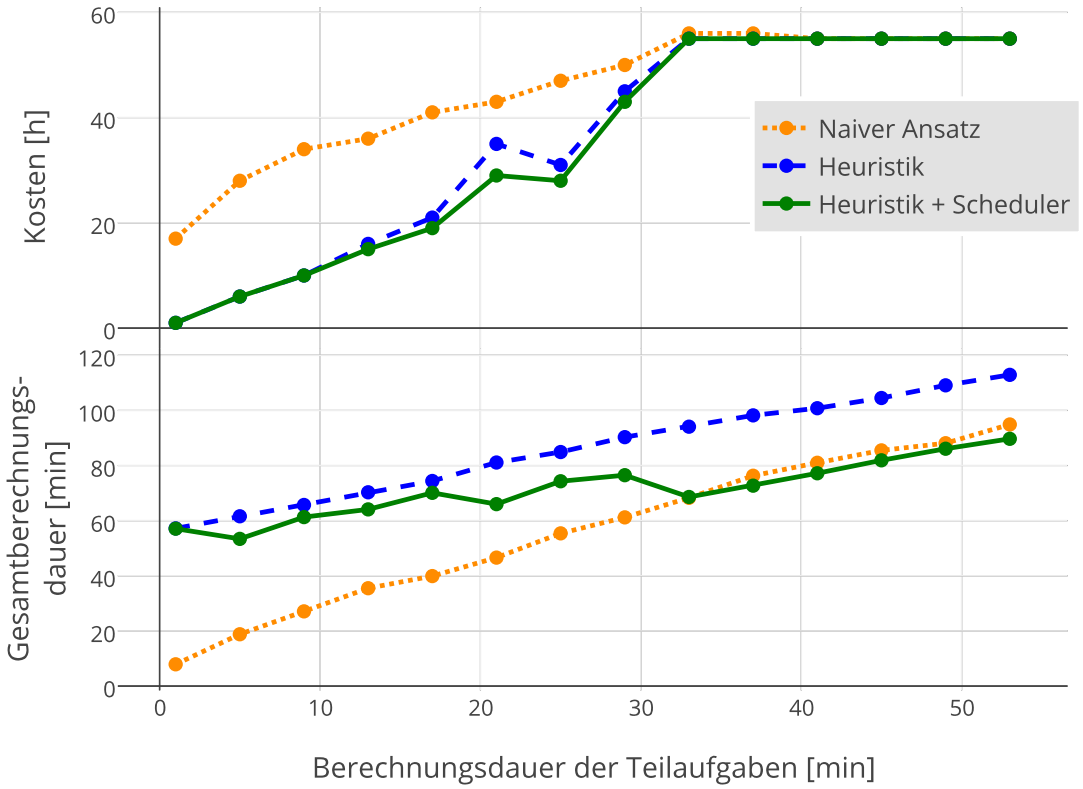

ABBILDUNG 3.5: ***Vergleich der Ansätze zur Bestimmung der Menge an Rechenknoten:*** *Gegenüberstellung der drei Ansätze um die Menge zusätzlich zu startender Rechenknoten abzuschätzen für 55 Einzelaufgaben. Der Ansatz mit Heuristik und Scheduling-Algorithmus benötigt für Berechnungen mit kurze Teilaufgaben weniger Ressourcen als der naive Ansatz und kann das Gesamtberechnungsergebnis für Berechnungen mit langen Teilaufgaben schneller liefern, da er die Rechenknoten früher startet.*

als der halben Abrechnungseinheit (i.e. eine halbe Stunde) haben, die Berechnungskosten zum Teil auf einen Bruchteil senken können. Bei einer Berechnungsdauer der Teilaufgaben von mehr als der halben Abrechnungseinheit entstehen mit allen Verfahren die gleichen Kosten und die Heuristik mit Scheduling-Algorithmus erzielt eine geringere Gesamtberechnungsdauer als der naive Ansatz.

Werden pro Maschine mehrere Berechnungsaufgaben gleichzeitig bearbeitet (siehe Kapitel 2 *Anforderungen an autonom skalierende Systeme*) muss darauf geachtet werden, dass bei Lastverringerung Maschinen frei werden, um diese herunterfahren zu können. Dazu dürfen die Maschinen nicht alle gleich stark ausgelastet werden. Andererseits nutzt eine Gleichauslastung die Berechnungskapazität der Maschinen optimal aus, besonders im Fall von Underprovisioning. Um beiden Anforderungen gerecht zu werden, werden die Berechnungsaufgaben auf einen Teil der Maschinen gleichmäßig verteilt. Im anderen Teil wird die Anzahl der freien Maschinen maximiert.

Eine weitere Anpassung an die stundenweise Abrechnung ist, dass nicht benötigte Maschinen erst kurz vor dem Erreichen der vollen Stunde heruntergefahren werden. Bis dahin laufen

sie für den Fall einer Laststeigerung im System weiter ohne Berechnungen durchzuführen.

3.1.2 Umsetzung Security

Um Sicherheitsanforderungen (siehe Abschnitt 2.1 *Security*) zu erfüllen wurden folgende Maßnahmen getroffen:

- Die Netzwerkverbindungen zwischen der Client-Applikation und dem Server sowie zwischen Server und Berechnungsknoten sind SSL-Verschlüsselt.
- Die Client-Applikation speichert den öffentliche Schlüssel des Servers um Man-in-the-Middle-Angriffe zu verhindern.
- Automatisch gestarteten Berechnungsknoten wird über die API des IaaS-Cloud-Providers der öffentliche Schlüssel des Servers und ein Einmalpasswort mitgegeben.
- Die Client-Applikation muss sich gegenüber dem Server mittels BenutzerInnenname und Passwort authentifizieren.
- Der Berechnungsknoten muss sich gegenüber dem Server entweder mittels BenutzerInnenname und Passwort, Einmalpasswort oder mittels Seriennummer und Token (siehe Unterunterabschnitt 2.1.1.5 *Defekte Authentifizierung und Sitzungsmanagement (Broken Authentication and Session Management)*) authentifizieren.

Da vertrauliche Informationen, welche zum Verbindungsaufbau benötigt werden, über die API des IaaS-Cloud-Providers ausgetauscht werden, ist das Abfangen von übertragenen Daten durch aktive Angriffe des IaaS-Cloud-Providers möglich. Dies verschlechtert die Sicherheit des Systems allerdings nicht, da der IaaS-Cloud-Provider ohnehin die Möglichkeit hat, Angriffe auf der Ebene der virtualisierten Hardware durchzuführen. Gegen passive Angriffe des IaaS-Cloud-Providers ist das System hingegen sicher, da nur Informationen zur Authentifizierung und keine Verschlüsselungsinformationen ersichtlich sind. Aktive Angriffe von außenstehenden Parteien werden ebenfalls verhindert, da die API-Aufrufe des IaaS-Cloud-Providers ebenfalls SSL-Verschlüsselt sind.

3.1.3 Umsetzung Monitoring und Kundenabrechnung

Monitoring und Kundenabrechnung wurden mit Hilfe eines gemeinsamen eventbasierten Systems umgesetzt. Dabei wird jedes Mal wenn sich der Status einer Berechnungsaufgabe oder eines Berechnungsknotens ändert ein Event erzeugt und gemeinsam mit Daten, die für Monitoring, Benchmarks und Kundenabrechnung benötigt werden, gespeichert.

Zusätzlich werden in periodischen Abständen Kennwerte des Systems ermittelt. Diese Kennwerte könnten zwar auch im Nachhinein aus den Event-Daten bestimmt werden, die periodische Ermittlung ermöglicht jedoch einen ständigen Überblick über den Betriebszustand des Netzwerks. Dazu werden die Kennwerte mittels Metrics [15] erfasst und können so auch in System-Monitor-Tools wie Ganglia [16] and Graphite [17] eingebunden werden.

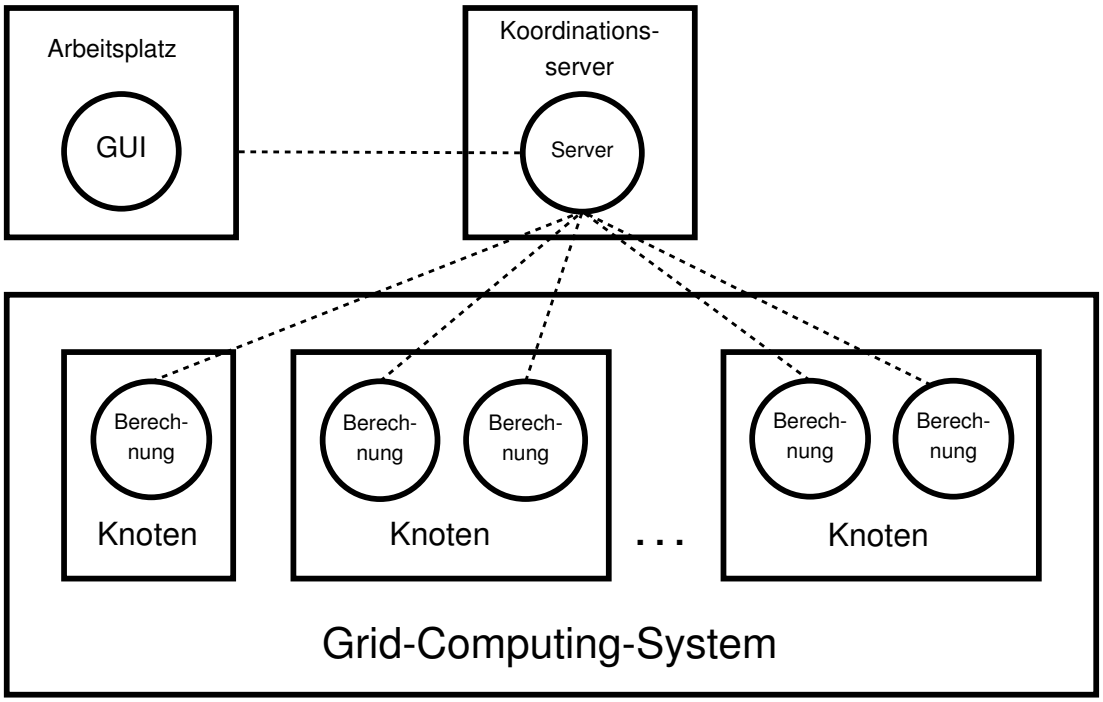

ABBILDUNG 3.6: ***Aufbau des Berechnungsnetzwerks auf Basis eines Grid-Computing-Systems:*** *Die Grid-Computing-Anbindung startet Instanzen der Berechnungsknotensoftware auf den Knoten des Grid-Computing-Systems. Die Berechnungsknotensoftware führt dann angeforderte Teilberechnungen durch. Alternativ kann auch der zentrale Koordinationsserver am Grid-Computing-System gestartet werden. Er wird dann von der grafischen BenutzerInnenoberfläche (GUI) mit Hilfe einer Datei auf einem verteilten Dateisystem gefunden.*

3.1.4 Implementierung

Um zusätzliche Rechenknoten in der IaaS-Cloud zu starten, wird die API des IaaS-Cloud-Providers benutzt um weitere Maschinen anzufordern. Diese sind so konfiguriert, dass ein vorher bereitgestelltes Festplattenabbild, auf dem die nötige Software vorinstalliert ist, gebootet wird. Außerdem wird über die API eine Zeichenkette an die Maschine übergeben. Diese wird nach dem Hochfahren an die Berechnungsknotensoftware weitergegeben und enthält alle Konfigurationen wie IP-Adresse sowie Port des zentralen Koordinationsserver und dessen öffentlichen Schlüssel. Mit Hilfe dieser Informationen verbindet sich die Berechnungsknotensoftware zum Koordinationsserver um eine Berechnungsaufgabe zugeteilt zu bekommen.

3.2 Grid-Computing-Anbindung

Bei der Implementierung dieses Prototyps wurde das bestehende Batch-Queuing-System um eine Anbindung an Sun Grid Engine [1] erweitert. Dadurch kann die Anzahl der vom Batch-Queuing-System verwalteten Rechenknoten autonom skaliert werden indem neue Instanzen der

Berechnungsknotensoftware am Grid-Computing-System gestartet bzw. beendet werden (siehe Abbildung 3.6). Das Batch-Queuing-System kann also nicht unbeschränkt sondern nur im Rahmen der vom Grid-Computing-System verwalteten Hardware skalieren.

Die Anbindung dient als Abstraktionsschicht zwischen Software, welche mit dem bestehende Batch-Queuing-System zusammenarbeiten kann und der Sun Grid Engine (siehe Unterabschnitt 3.2.1 *Grundlegende Architektur*). Weiters ermöglicht sie das Nutzen der Ressourcen des Grid-Computing-Systems als eine von mehreren Rechenknotengruppen (siehe Kapitel 3 *Methoden für verteiltes Rechnen*).

3.2.1 Grundlegende Architektur

Um die Anbindung möglichst Grid-Computing-System unabhängig zu implementieren, werden zwei Adaptionsschichten eingezogen:

- Funktionen des Grid-Computing-Systems werden nicht direkt aufgerufen sondern immer über Scripts angesprochen. Durch Anpassungen an diesen Scripts ist es möglich neue Grid-Computing-Systeme anzusprechen ohne etwas an der Anwendungslogik zu ändern.

- Die Berechnungsaufgaben werden nicht direkt am Grid-Computing-System gestartet. Stattdessen wird eine Instanz der Berechnungsknotensoftware gestartet, welche eine Netzwerkverbindungen zum zentrale Koordinationsserver aufbaut. Die Berechnungsknotensoftware sorgt für das Übertragen von Eingabedateien der eigentlichen Berechnungssoftware und dafür, dass diese gestartet wird. Außerdem kann sie Ausgabedateien und Textausgaben schon vor Ende der Berechnung an die Benutzerin oder den Benutzer senden um diesem möglichst früh eine Rückmeldung zu geben.

- Je nach Konfigurationen beendet sich die Berechnungsknotensoftware sofort nach der Berechnung einer einzelnen Teilaufgabe oder erst wenn keine Teilaufgaben mehr bearbeitet werden und auch keine für diesen Berechnungsknoten passenden Teilaufgaben am zentralen Koordinationsserver auf die Berechnung warten. Je nach Einsatzgebiet und Unternehmenskultur kann die passende Konfigurationen gewählt werden. Bei Berechnungsaufgabe mit vielen sehr kurzen Teilaufgaben verbessert das Verwenden einer Berechnungsknotensoftwareinstanz für mehrere Teilaufgaben (keep alive) erheblich. Bei Berechnungen mit langer Gesamtberechnungsdauer kann dieses Verhalten jedoch als ungerecht empfunden werden, da sich die Benutzerin oder der Benutzer für mehrere Teilaufgaben nur einmal in der Warteschlange des Grid-Computing-Systems anstellen muss und erhaltene Ressourcen bis zum Ende der Berechnung nicht wieder frei gibt. Dadurch blockiert ein Nutzer unter Umständen einen großen Teil der Hardware mit einer Berechnung und andere Nutzer, mit unter Umständen deutlich kürzeren Berechnungen, kommen nicht zum Zug. In diesem Fall sollte auf die “keep alive”-Option verzichtet werden, da der Performanceunterschied bei Teilaufgaben mit langer Berechnungsdauer vernachlässigbar ist (siehe Abschnitt 4.1 *Performancevergleich anhand von synthetischen Beispielen*).

3.2.2 Umsetzung Security

Die Verbindung zwischen zentralem Koordinationsserver und Berechnungsknotensoftware sind mit der gleichen Verschlüsslung wie bei IaaS-Cloud-Anbindung (siehe Unterabschnitt 3.1.2 *Umsetzung Security*) abgesichert. In der Implementierung für diese Anbindung erfolgt der Austausch des Einmalpassworts über das Grid-Computing-System. Je nach Wahl des Grid-Computing-Systems und dessen Konfigurationen kann diese Information jedoch von anderen Nutzern des Systems eingesehen werden.

Alternativ könnte das Einmalpasswort in einem verteilten Dateisystem mit Zugriffskontrolle abgelegt werden um den Zugriff Unbefugter zu verhindern. Da Grid-Computing-Systeme aber meinst firmen- oder institutsintern eingesetzt werden und der Zugriff von außen nicht möglich ist, sind Sicherheitsmaßnahmen dieser Art nur in speziellen Anwendungsszenarien nötig.

3.2.3 Umsetzung Monitoring und Kundenabrechnung

Monitoring und Kundenabrechnung wurden mit Hilfe des gleichen eventbasierten Systems umgesetzt wie bei der IaaS-Cloud-Anbindung (siehe Unterabschnitt 3.1.3 *Umsetzung Monitoring und Kundenabrechnung*).

KAPITEL 4

Vergleich der Anbindungen

In diesem Kapitel werden die entwickelten Anbindungen untersucht. Dazu werden sie miteinander, mit vorab manuell gestarteten Rechenknoten und mit einem Hybridsystem, das sowohl Grid-Computing-Anbindung als auch IaaS-Cloud-Anbindung verwendet, verglichen.

4.1 Performancevergleich anhand von synthetischen Beispielen

Aufgrund der, im Gegensatz zu Grid-Computing-Systemen, scheinbar unbegrenzten Ressourcen der IaaS-Cloud ist der direkte Vergleich der Zeit bis zum Abschluss einer Berechnungsaufgabe alleine nicht aussagekräftig. Solange die Anzahl der Teilaufgaben groß genug ist, ist die Ausführung auf der IaaS-Cloud dank des Ressourcenvorteils immer schneller, wenn man nicht auf stundenweise Abrechnung optimiert. Optimiert man andererseits auf stundenweise Abrechnung (siehe Unterabschnitt 3.1.1 *Grundlegende Architektur*) und vergleicht das Ergebnis mit einem Grid-Computing-System das über ausreichend Ressourcen verfügt um alle Teilaufgaben parallel auszuführen, ist klarerweise die Ausführung am Grid-Computing-System schneller, da hier der Overhead der Berücksichtigung des Abrechnungsmodells wegfällt.

Daher wird im Folgenden die Geschwindigkeit der implementierten Anbindungen ohne den Einfluss von scheinbar unbegrenzten Ressourcen und Optimierung auf stundenweise Abrechnung analysiert. Dazu wird die gleiche Berechnungsaufgaben mit 320 simulierten Teilaufgaben mit vorgegebener Berechnungsdauer unter Verwendung der beiden Anbindungen bzw. schon vorab manuell gestarteter Berechnungsknotensoftware abgearbeitet. Die zur Verfügung stehenden Ressourcen werden dabei jeweils zwischen 8 und 32 parallel nutzbaren Berechnungsslots variiert. Dabei kann sowohl zwischen den Anbindungen verglichen werden, als auch die Skalierbarkeit einer einzelnen Anbindung bei Erhöhung der Hardwaremenge untersucht werden.

Bei dem Vergleich der Gesamtberechnungsdauer von 320 simulierten Teilaufgaben mit jeweils 10 Sekunden Berechnungsdauer (siehe Abbildung 4.1) zeigt sich, dass die bisherige

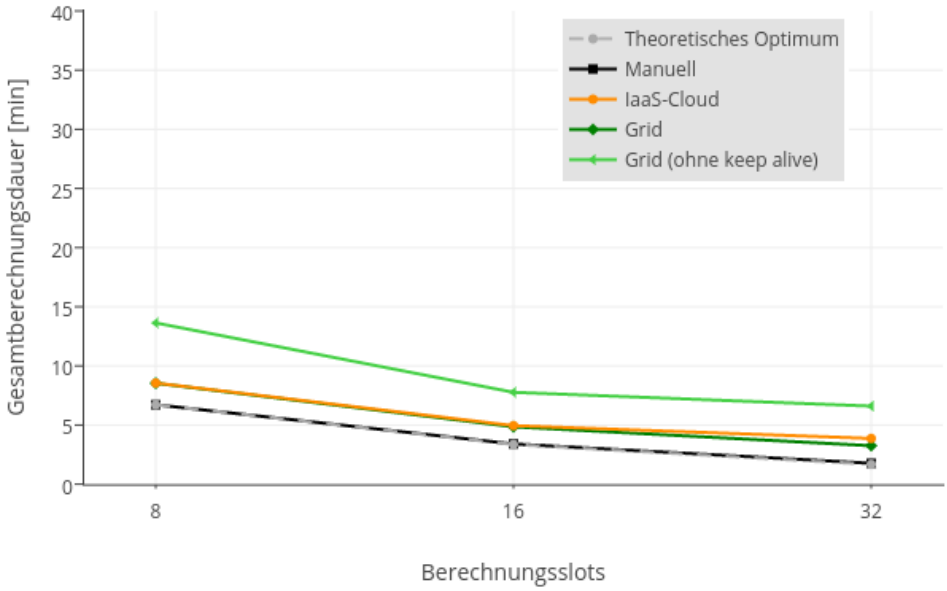

ABBILDUNG 4.1: *Vergleich der Gesamtberechnungsdauer einer Berechnungsaufgabe mit 320 simulierten Teilaufgaben mit jeweils 10 Sekunden Berechnungsdauer bei Verwendung der unterschiedlichen Anbindungen. Die zur Verfügung stehende Hardwaremenge wird dabei zwischen 8 und 32 parallel nutzbaren Berechnungsslots variiert.*

Lösung der manuell vorab gestarteten Berechnungsknotensoftware bei der Skalierung der Ressourcenmenge beinahe an das theoretische Optimum herankommt. Das heißt, bei doppelter Hardwaremenge reduziert sich die Berechnungsdauer auf die Hälfte, bei vierfacher Hardwaremenge auf ein Viertel.

Bei Nutzung der IaaS-Cloud werden die benötigen Recheninstanzen erst bei Bedarf, also nach Starten der Gesamtberechnung angefordert und hochgefahren. Dadurch verlängert sich die Gesamtberechnungsdauer entsprechend um die Zeit, die für Anforderung, Hochfahren der Maschinen und Initialisieren der Berechnungsknotensoftware benötigt wird.

Bei Verwendung der Grid-Computing-Anbindung werden die Ressourcen ebenfalls erst bei Bedarf angefordert. Hier müssen die Maschinen zwar nicht mehr hochgefahren werden, jedoch hat das Grid-Computing-System ein fixes Scheduling-Intervall. Durch die Wartezeit bis zum nächsten Scheduling-Durchlauf entsteht eine zusätzliche Verzögerung. Außerdem werden die Berechnungsslots einzeln vom Grid-Computing-System angefordert und pro Berechnungsslot

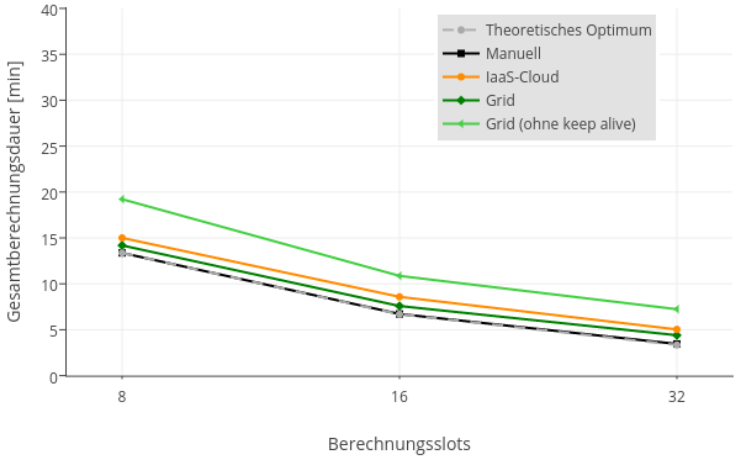

ABBILDUNG 4.2: *Vergleich der Gesamtberechnungsdauer einer Berechnungsaufgabe mit 320 simulierten Teilaufgaben mit jeweils 20 Sekunden Berechnungsdauer bei Verwendung der unterschiedlichen Anbindungen. Die zur Verfügung stehende Hardwaremenge wird dabei zwischen 8 und 32 parallel nutzbaren Berechnungsslots variiert.*

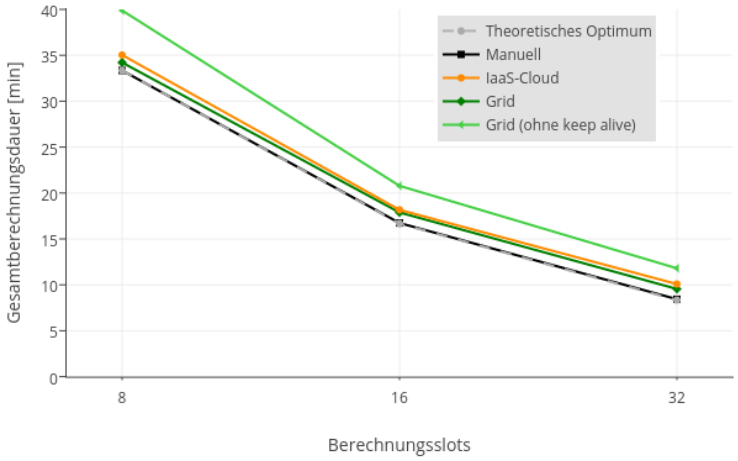

ABBILDUNG 4.3: *Vergleich der Gesamtberechnungsdauer einer Berechnungsaufgabe mit 320 simulierten Teilaufgaben mit jeweils 50 Sekunden Berechnungsdauer bei Verwendung der unterschiedlichen Anbindungen. Die zur Verfügung stehende Hardwaremenge wird dabei zwischen 8 und 32 parallel nutzbaren Berechnungsslots variiert.*

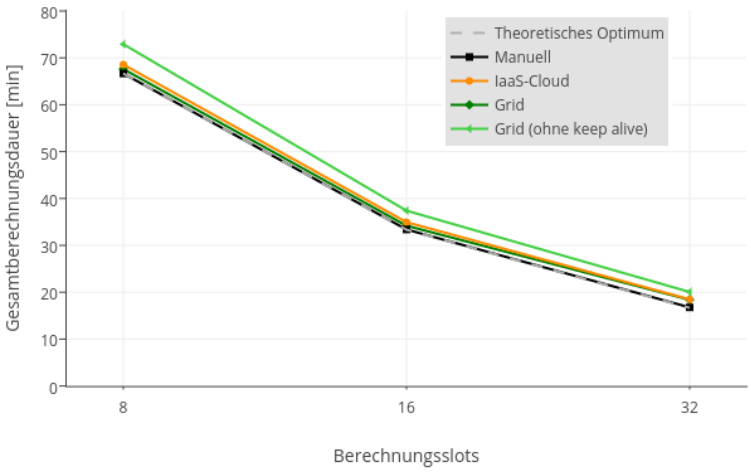

ABBILDUNG 4.4: *Vergleich der Gesamtberechnungsdauer einer Berechnungsaufgabe mit 320 simulierten Teilaufgaben mit jeweils 100 Sekunden Berechnungsdauer bei Verwendung der unterschiedlichen Anbindungen. Die zur Verfügung stehende Hardwaremenge wird dabei zwischen 8 und 32 parallel nutzbaren Berechnungsslots variiert.*

eine Instanz der Berechnungsknotensoftware gestartet. Im Vergleich dazu wird von der IaaS-Cloud immer eine Maschine für mehrere, bei diesem Benchmark acht, Berechnungsslots angefordert und pro Maschine nur eine Instanz der Berechnungsknotensoftware ausgeführt. Ein Teil des Overheads ist also auch auf die zusätzliche Belastung des zentralen Koordinationsservers durch die acht mal so große Anzahl an Berechnungsknotensoftwareinstanzen zurückzuführen. Um diesen Nachteil auszugleichen ist es denkbar von der Grid-Computing-Anbindung mehrere Berechnungsslots gleichzeitig anzufordern. Dies kann allerdings dazu führen, dass Berechnungsslots unnötig lange blockiert werden, wenn die Teilaufgaben in den Berechnungsslots unterschiedlich viel Zeit in Anspruch nehmen.

Wird die “keep alive”-Option für die Grid-Computing-Anbindung ausgeschaltet (siehe Unterabschnitt 3.2.1 *Grundlegende Architektur*), so berechnet jede Instanz der Berechnungsknotensoftware eine Teilaufgabe und beendet sich dann, um Fairness der Ressourcenverteilung sicherzustellen. Dadurch entsteht der Overhead durch Grid-Computing-System und Initialisieren der Berechnungsknotensoftware für diesen Benchmark nicht 32 sondern 320 mal und verlängert die Gesamtberechnungsdauer entsprechend.

Bei Verlängerung der Berechnungsdauer der Teilaufgaben auf 20 (siehe Abbildung 4.2), 50 (siehe Abbildung 4.3) bzw. 100 Sekunden (siehe Abbildung 4.4) zeigt sich, dass der Overhead hier bei der Gesamtberechnungsdauer immer weniger ins Gewicht fällt.

Bei einer Analyse des Overheads an Berechnungsdauer gegenüber der Berechnung auf

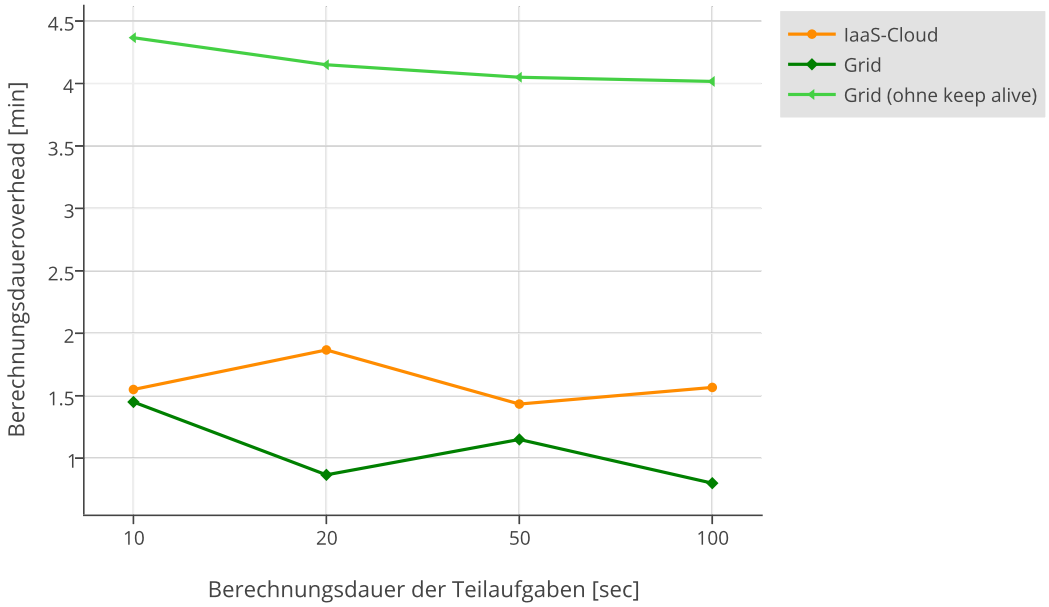

ABBILDUNG 4.5: *Analyse des Overheads an Berechnungsdauer gegenüber der Berechnung auf vorab manuell gestarteter Berechnungsknotensoftware bei Verwendung von 16 Berechnungsslots. Die Auswertung zeigt, dass der Berechnungsdauer-Overhead von der Berechnungsdauer der Teilaufgaben unabhängig ist.*

vorab manuell gestarteter Berechnungsknotensoftware (siehe Abbildung 4.5) zeigt sich, dass der Overhead nicht von der Berechnungsdauer der Teilaufgaben abhängig ist. Da bei IaaS-Cloud-Anbindung und Grid-Computing-Anbindung mit “keep alive”-Option alle Maschinen bzw. Berechnungsknotensoftwareinstanzen zu Beginn angefordert werden und dann dann eine Teilaufgabe nach der anderen abarbeiten, ist der Overhead hier also auch nicht von der Anzahl der Teilaufgaben abhängig. Hier fällt wird der Overhead also umso weniger ins Gewicht, je länger die Gesamtberechnungsdauer ist, unabhängig davon ob diese durch lange oder viele Teilaufgaben zustande kommt. Wird die Grid-Computing-Anbindung ohne “keep alive”-Option, verwendet so sind Teilaufgaben mit langer Berechnungsdauer optimal. Durch eine hohe Anzahl an Teilaufgaben erhöht sich in diesem Fall der Overhead.

Da Simulation von Nano-Bauelementen meist mindestens einige Minuten, oft sogar mehrere Stunden bis Tage in Anspruch nehmen, ist der Overhead für diesen Anwendungsbereich also vernachlässigbar klein.

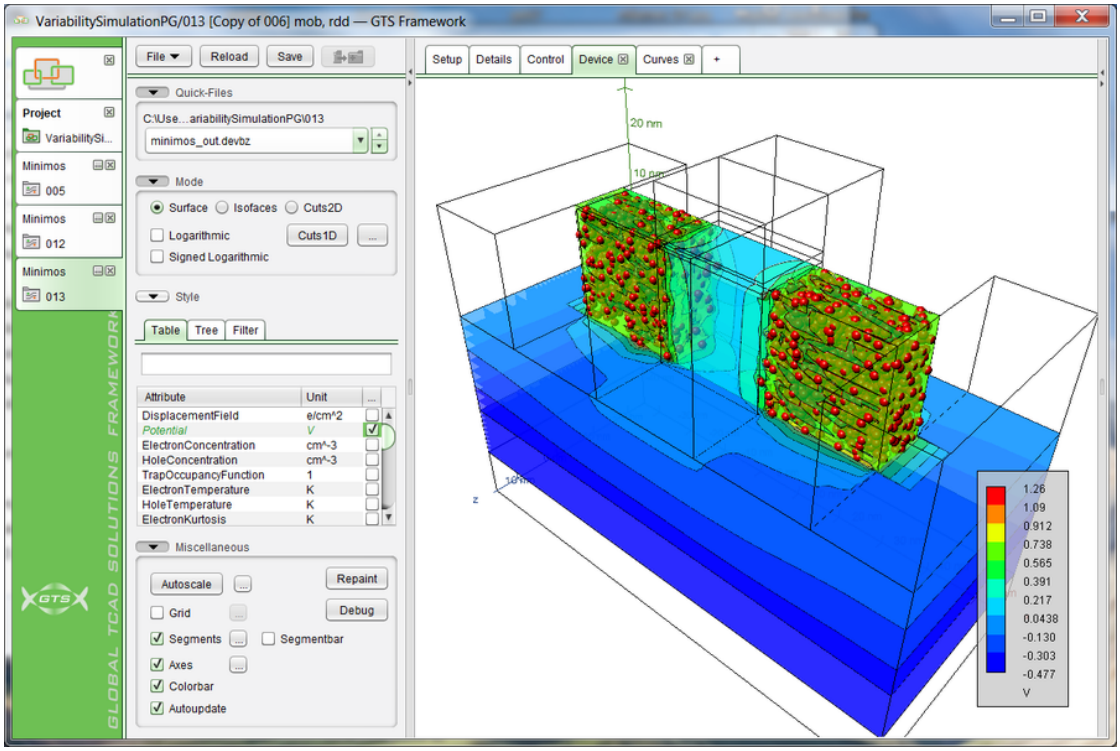

ABBILDUNG 4.6: *Virtuelles Bauelement mit zufällig verteilten, diskreten Dotieratomen (Quelle: [18])*

4.2 Einsatz im Nano-TCAD

Im Zuge dieser Arbeit soll die Problemstellung, einem Batch-Queuing-System die Fähigkeit zur autonomen Skalierung zu geben, anhand des „Global TCAD Solutions Framework“ [18] (kurz GTS-Framework) beleuchtet werden. Das GTS-Framework ist eine Umgebung für die numerische Simulation des Herstellungsprozesses und des elektrischen Verhaltens von Halbleiterbauelementen. Neuartige Bauelementstrukturen werden in einem Computermodell abgebildet, um so das Verhalten vorhersagen zu können (siehe Abbildung 4.6). Das GTS-Framework stellt Simulationstools sowie grundlegende Funktionalität für die Erstellung von Bauelementen und zur Visualisierung von Simulationsergebnissen zur Verfügung. Zum Strukturieren der Simulationsaufgaben ist eine Projektverwaltung verfügbar.

Mit dem GTS-Framework kann der gesamte Simulationsprozess über ein gemeinsames Frontend mit graphischer BenutzerInnenoberfläche (GUI) gesteuert werden. Das Framework umfasst:

- Strukturgenerierung
- Definition der Simulationsaufgabe
- Start des Simulationsprozesses
- Visualisierung der Ergebnisse

Schwerpunkt wird auf die Simulation von Bauelementen mit Abmessungen im Nanometer-Bereich, so genannte Nano-Bauelemente, gelegt.

Das GTS-Framework wird von führenden Halbleiterherstellern eingesetzt um die Entwicklungszyklen von neuartigen Bauelementen zu verkürzen. Dafür sind rechenintensive Simulationen notwendig, welche bisher nur auf Großrechnern durchgeführt werden konnten. Im Zuge meiner Bachelorarbeit [19] habe ich das User-Interface von der Projektverwaltung und der Simulationskomponente entkoppelt, um effizienteres Arbeiten in der Entwicklung neuer Bauelemente zu ermöglichen.

Teil des GTS-Frameworks ist auch ein Batch-Queuing-System, welches im Zuge einer Diplomarbeit [3] entwickelt wurde. Dieses nutzt bisher allerdings nur manuell gestartete Rechenknoten und besitzt nicht die Fähigkeit sich autonom an Lastschwankungen anzupassen.

Durch die fortschreitende Verkleinerung der Bauelemente sind einige Annahmen, die bisher in der Bauelementsimulation getroffen wurden, nicht mehr exakt genug. So wurde beispielsweise in dotierten Bereichen eines Halbleiterbauelements eine gleichmäßige Verteilung der Dotieratome angenommen. Durch die geringere Zahl der Atome in kleineren Bauelementen beeinflusst jedoch die genaue Position jedes Dotieratoms das Gesamtverhalten des Bauteils signifikant [20]. Ein solches Bauelement wird in Abbildung 4.6 gezeigt und in Unterabschnitt 4.2.1 (*Unabhängige Simulationen von Nano-Bauelementen*) simuliert. Daher ist bei diesen Bauelementen nicht eine Simulation ausreichend, wie bisher, sondern es werden mehrere hundert virtuelle Bauelemente mit zufälliger Atomverteilung generiert und anschließend simuliert. Dies ist nur eines von vielen Beispielen im Bereich der Simulation von Nano-Bauelementen, bei denen viele hundert Simulationen notwendig sind. Andere Beispiele sind das Simulieren des Einflusses von Unebenheiten an den Oberflächen des Bauelements auf dessen Eigenschaften, das Simulieren von Bauelementen auf der Grundlage von quantenmechanischen Modellen [21] [22] und die Analyse des “bias temperature instability (BTI)”-Effekts eines PMOS-Transistors [23] (siehe Unterabschnitt 4.2.2 *Simulationsketten zur Analyse von Nano-Bauelementen*).

Diese mehreren hundert Simulationen sind völlig unabhängig voneinander und können daher nicht nur wie bisher nacheinander auf einer einzigen oder einer festen Anzahl von Maschinen abgearbeitet werden, sondern auch parallel. In der Vergangenheit hätte die massive parallele Bearbeitung zu hohen, fixen Kosten geführt, da enorme Mengen an Hardware angeschafft hätten werden müssen. Doch durch die Entwicklungen im Bereich Cloud-Computing ist es heute möglich bei gleichen Kosten wenige Maschinen für lange Zeit oder viele Maschinen für kurze Zeit zu mieten. Dadurch kann die Entwicklung von Bauelementen beschleunigt werden, ohne dass zusätzliche Kosten entstehen. Diesen Technologievorteil wollen wir auch mit dem GTS-Framework nutzbar machen.

In der derzeitigen Architektur (siehe Abbildung 4.7) sind User-Interface, Projektverwaltung und Simulation bereits getrennt und die Ausführung der Einzelkomponenten auf verschiedenen Maschinen ist möglich. Daher muss an der Architektur nichts verändert werden und die Änderungen finden hauptsächlich in der Komponente zur Simulation statt. Um Steuerung und Überwachung der Simulation durch den Anwender zu ermöglichen und um ein Billing-System betreiben zu können, müssen aber auch die anderen Komponenten angepasst werden.

ABBILDUNG 4.7: *Softwarearchitektur von BenutzerInnenoberfläche und Berechnungsnetzwerk*

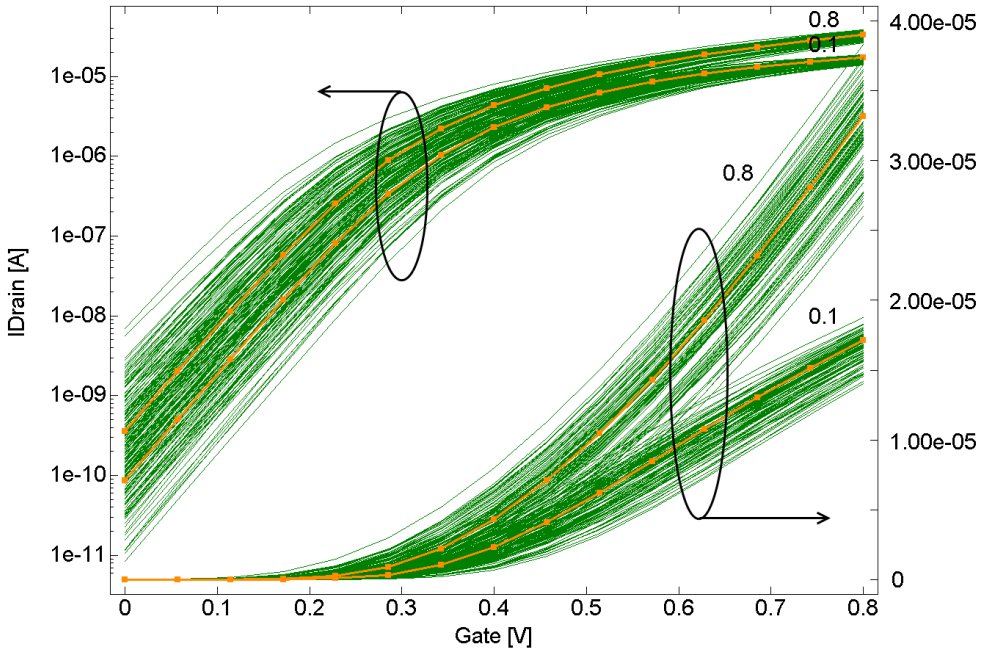

ABBILDUNG 4.8: *Drain-Strom-Kennlinien der ersten 100 von 480 zufälligen Dotieratomverteilungen für $V_{Drain} = 0.1V$ bzw. $V_{Drain} = 0.8V$. Der Drain-Strom wird im Diagramm einmal logarithmisch (links-oben) aufgetragen um den Verlauf für kleine Gate-Spannungen darzustellen und einmal linear (rechts-unten) aufgetragen um den annähernd linearen Bereich ab etwa $0.5V$ zu zeigen. Die orangen Linien und Kästchen zeigen die Berechnungsergebnisse für den Nominal-Fall mit kontinuierlichem Dotierprofil, also ohne diskrete Positionen der einzelnen Dotieratome.*

4.2.1 Unabhängige Simulationen von Nano-Bauelementen

In diesem Abschnitt wird die Performance der Anbindungen anhand des Beispiels einer Simulation eines Nano-Bauelements verglichen. Da der simulierte Transistor Abmessungen von wenigen Nanometern hat, in diesem Beispiel etwa 20 nm, ist die Annahme, dass die Dotieratome in den dotierten Bereichen des Halbleiterbauelements gleichmäßig verteilt sind, nicht zulässig [20]. Die Postionen der einzelnen Dotieratome hat entscheidenden Einfluss auf die Eigenschaften des Transistors. Daher werden auf Basis des Transistors mit gleichmäßig verteilten Dotieratomen mehrere hundert virtuelle Bauelemente mit zufälliger Atomverteilung generiert und anschließend simuliert. Eines dieser Bauelemente wird in Abbildung 4.6 gezeigt. Das Ergebnis jeder Einzelsimulation besteht aus den zwei Kennwerten der Schwellenspannung $V_{th,lin}$ und $V_{th,sat}$ und zwei Kurven, jeweils für den linearen und für den gesättigten Bereich (siehe Abbildung 4.8.

Die Kennwerte beschreiben vereinfacht gesagt den Punkt ab dem der Transistor als eingeschaltet betrachtet wird. Bei der Analyse besonders interessant ist die Abweichung von $V_{th,lin}$ und $V_{th,sat}$ bei verschiedenen Dotieratomverteilungen. Bei der Produktion in großen Stückzahlen sollen die Abweichungen von $V_{th,lin}$ und $V_{th,sat}$ in definierten Bereichen liegen und dabei die Menge an Ausschuss, der diese Kriterien nicht erfüllt, minimiert werden. Dies ist wichtig, da beispielsweise Verstärker- und Inverter-Schaltungen zwei Transistoren mit möglichst gleichem

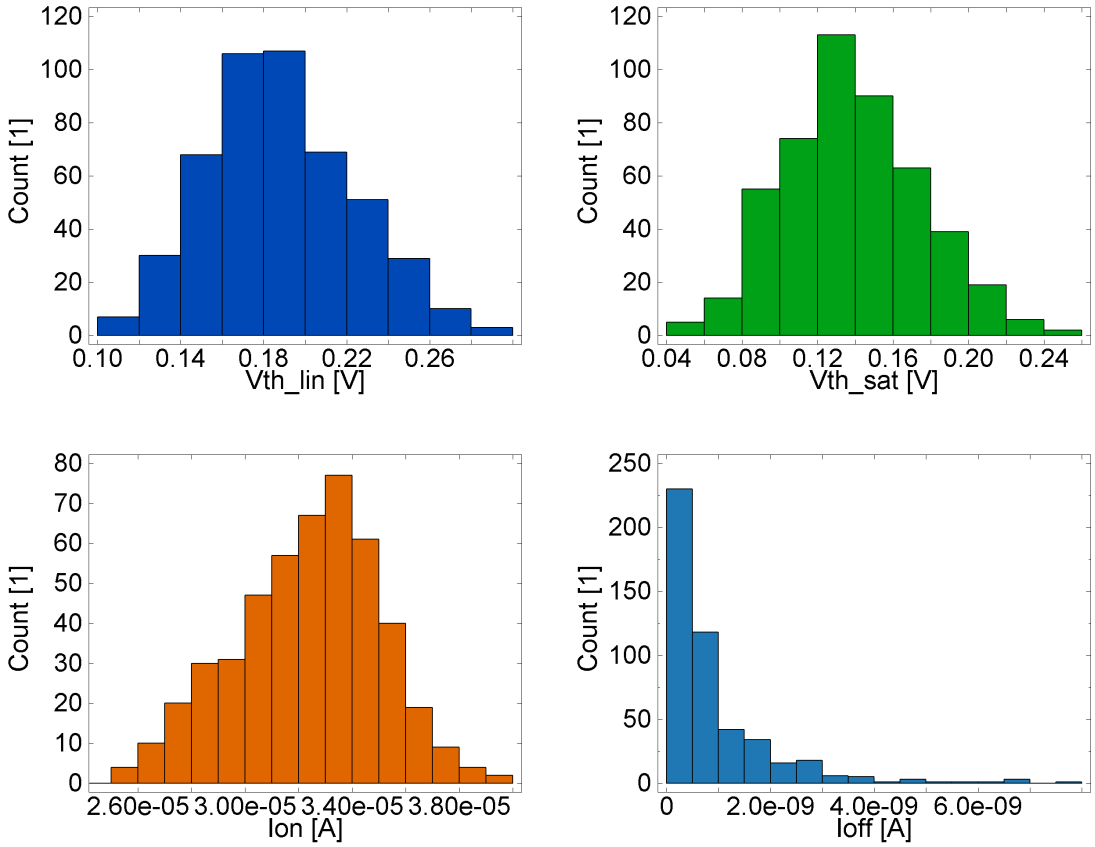

ABBILDUNG 4.9: *Histogramme zur Darstellung der Verteilung der Kennwerte $V_{th,lin}$, $V_{th,sat}$, I_{on} und I_{off} für 480 zufälligen Dotieratomverteilungen. Dabei wird der Schwankungsbereich der Kennwerte durch den Einfluss der Positionen der Dotieratome deutlich.*

Verhalten benötigen. Diese Kennwerte aller Einzelsimulation werden dann zusammengefasst um eine statistische Auswertung zu machen (siehe Abbildung 4.9.

Um eine aussagekräftige statistische Auswertung machen zu können, muss die Anzahl der Stichproben, je nach Stärke des Einflusses der Dotieratomverteilung auf die Kennwerte, ausreichend groß sein. Der Einfluss der Dotieratomverteilung ist umso größer, je kleiner das Bauelement ist, hängt aber auch von anderen Faktoren wie Bauform und Materialien ab. Für die statistische Analyse des Transistors in diesem Beispiel sind einige hundert Simulationen mit unterschiedlichen Dotieratomverteilungen nötig.

Für die Berechnung dieses Transistors werden 480 unterschiedliche Dotieratomverteilungen simuliert. Dabei benötigten die Einzelberechnung auf den Grid-Maschinen zwischen 13.2 und 19.3 Minuten, durchschnittlich 17.6 Minuten (siehe Abbildung 4.10). Die Unterschiede in der Berechnungszeit kommen daher, dass sich die Zeit, die der Bauteilsimulator zum Konvergieren benötigt, schon bei leicht verschiedenen Bauteilkonfiguration ändert. Auf den Cloud-Instanzen benötigen dieselben Einzelberechnungen zwischen 21.7 und 28.5 Minuten, durchschnittlich 24.7

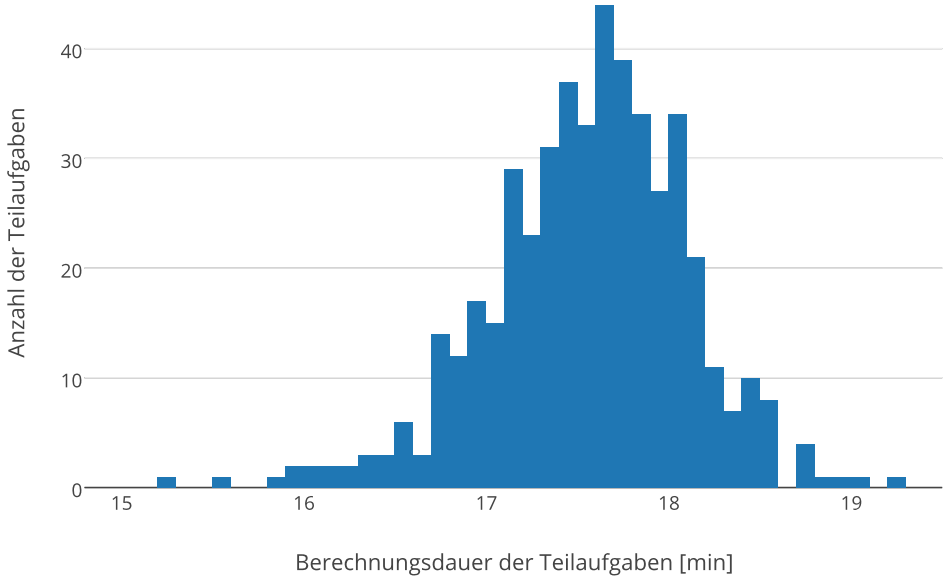

ABBILDUNG 4.10: *Berechnungsdauern der Simulationen mit unterschiedlichen Dotieratomverteilungen auf den Grid-Maschinen*

Minuten, da diese eine geringere Floating-Point-Performance besitzen. Die Ergebnisse sind daher nicht direkt vergleichbar, sondern müssen auf die durchschnittliche Einzelberechnungsdauer normiert werden. Aber auch die Betrachtung der nicht normierten Daten ist relevant um die Performance für den Einsatz in einem Unternehmen zu vergleichen. Die dort verwendeten Grid-Maschinen wurden für die Berechnung der für dieses Unternehmen typischen Aufgaben ausgelegt und können Einzelberechnungen daher in kürzerer Zeit durchführen als Universalrechner in einer IaaS-Cloud.

Die Ergebnisse der Berechnungen des Transistors unter Verwendung der verschiedenen Anbindungen sind in Tabelle 4.1 zu sehen. Bei der Berechnung mit vorab manuell gestarteter Berechnungsknotensoftware bzw. am Grid wurden dieselben Maschinen verwendet, die Ergebnisse sind also direkt vergleichbar. Es wurden hier jeweils zehn Maschinen mit bis zu acht parallelen Berechnungen verwendet. Dabei bestätigt sich die Beobachtung aus Abschnitt 4.1 (*Performancevergleich anhand von synthetischen Beispielen*), dass der Overhead der Grid-Anbindung vernachlässigbar klein ist.

Die Menge der benötigten Hardwareressourcen, die zur Ausführung der Simulation benötigt wird, ist für die Berechnung mit vorab manuell gestarteter Berechnungsknotensoftware allerdings minimal höher, als wenn diese am Grid gestartet wird (siehe Abbildung 4.11). Dieser

	10% fertig [min]	90% fertig [min]	Berechnungs-dauer [min]	Kosten
Manuell	18.2	105.8	107.7	17.9 Maschinenstunden (8 Berechnungsslots pro Maschine)
Grid	18.6	106.3	108.1	17.7 Grid-Maschinenstunden (8 Berechnungsslots pro Maschine)
IaaS-Cloud	31.5	64.4	69.3	72 Cloud-Instanz-Abrechnungseinheiten (4 Berechnungsslots pro Instanz)
Hybrid	21.3	58.4	62.4	6.2 Grid-Maschinenstunden 43 Cloud-Instanz-Abrechnungseinheiten

TABELLE 4.1: ***Berechnungsdauern und Kosten im Beispiel mit unabhängigen Simulationen:*** *Die Berechnung unter Verwendung des Hybridsystems liefert das Gesamtergebnis in kürzerer Zeit als die Berechnung in der IaaS-Cloud. Unter der Annahme, dass die Nutzung der Grid-Ressourcen weniger kostet als jene der IaaS-Cloud, verursacht die Nutzung des Hybridsystems auch weniger Kosten.*

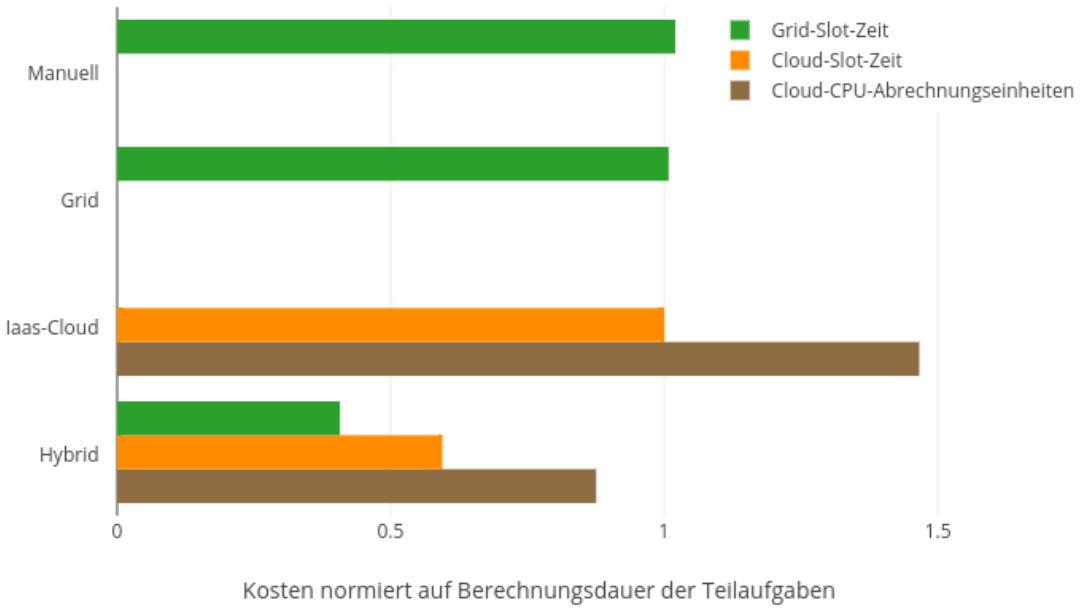

ABBILDUNG 4.11: *Menge der Hardwareressourcen die zur Ausführung der Simulation unter Verwendung der verschiedenen Anbindungen benötigt werden. Die Slot-Zeit ist dabei die Gesamtzeit für die Slots durch Dateitransfer, Kommunikations-Overhead oder die eigentliche Berechnung blockiert werden.*

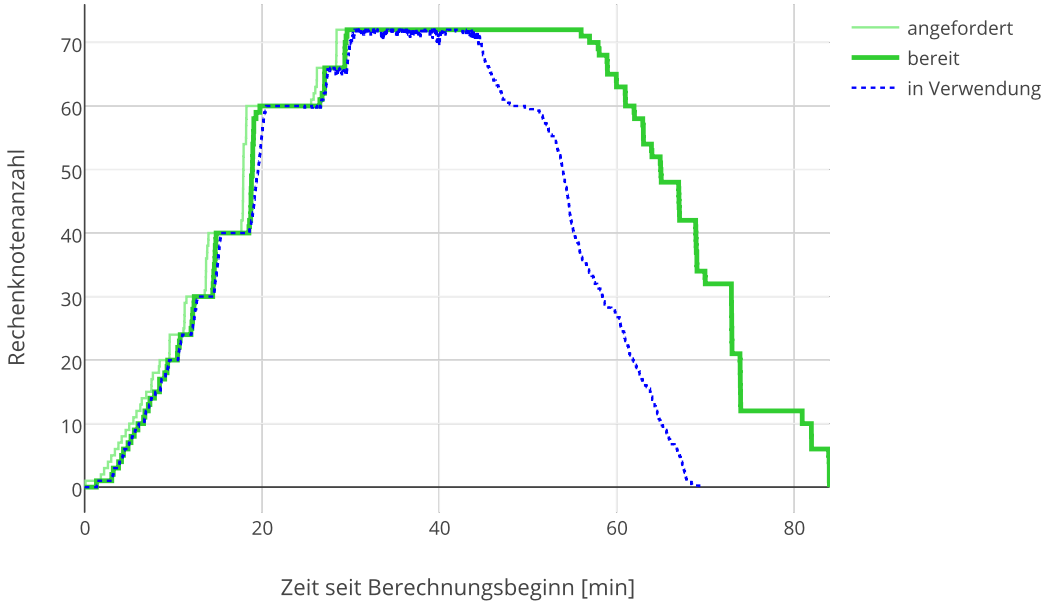

ABBILDUNG 4.12: ***Ablauf der Berechnung in der IaaS-Cloud:*** *Zeitlicher Ablauf der Berechnung von 480 unabhängigen Simulationen in der IaaS-Cloud unter Verwendung des heuristischen Ansatz mit Scheduling-Algorithmus zur Bestimmung der Menge an zu startenden Rechenknoten. Während der Hochlaufphase des Systems (0 bis 30 Minuten) wird je nach zu erwartender Mindestmenge an benötigter Rechenzeit schrittweise zusätzliche Hardware angefordert. Ab dem Zeitpunkt an dem keine weiteren unbearbeiteten Teilaufgaben mehr vorliegen (etwa Minute 45) befinden sich mehr und mehr Maschinen im Leerlauf. Diese werden allerdings nicht sofort heruntergefahren, sondern erst kurz vor Ende der Abrechnungseinheit.*

Effekt kommt daher, dass sich eine Instanz der Berechnungsknotensoftware am Grid sofort beendet sobald sie keine Berechnungen mehr durchführt, während die manuell gestarteten Berechnungsknotensoftwareinstanzen erst nach Fertigstellung aller Teilberechnungen manuell beendet werden (siehe Abbildung 4.13). Dies führt besonders bei großen Schwankungen der Teilberechnungsdauer zu unnötig lange blockierten Maschinen.

Bei der Berechnung unter Verwendung der IaaS-Cloud wurde der heuristische Ansatz mit Scheduling-Algorithmus zur Bestimmung der Menge an zu startenden Rechenknoten aus Unterabschnitt 3.1.1 (*Grundlegende Architektur*) verwendet (siehe Abbildung 4.12). Trotz des Nachteils, der sich durch das verzögerte Anfordern der Cloud-Instanzen gegenüber der Grid-Anbindung ergibt, liegt das Berechnungsergebnis nach nur 64% der Zeit vor.

Betrachtet man den Ressourcenbedarf normiert auf die reine Berechnungsdauer der Teilaufgaben, zeigt sich, dass der Overhead durch den Transfer der Eingabedateien und durch den

Kommunikationsaufwand in diesen Beispiel nicht ins Gewicht fällt. Die Zeit, in der die Hardware benutzt wird, ist sowohl bei der Berechnung am Grid, als auch bei der Berechnung in der Cloud, wo die Bandbreite zwischen zentralem Koordinationsserver und Berechnungsknoten geringer ist, um weniger als 1% länger als die reine Berechnungsdauer der Teilaufgabe. Bei Berechnungen mit größeren Eingabedateien würde sich dieser Overhead allerdings erhöhen.

Wird, um den Ressourcenbedarf in der IaaS-Cloud zu ermitteln, allerdings die Anzahl der Abrechnungseinheiten und nicht die Zeit, in der die Hardware benutzt wurde, betrachtet, so zeigt sich ein Overhead von über 46%. Der hohe Overhead in diesem Benchmark kommt daher, dass einige Teilaufgaben eine Berechnungsdauer von nahezu einer halben Abrechnungseinheit haben. Zusammen mit der Zeit zum Hochfahren des Rechenknotens und einer Zeitpauschale die für das Herunterfahren des Rechenknotens vor Ende der Abrechnungseinheit von der Heuristik zur Bestimmung der Menge an zu startenden Rechenknoten berücksichtigt wird, führt dies dazu, dass die Heuristik davon ausgeht das auf dem Rechenknoten keine zweite Teilaufgabe vor Ende der Abrechnungseinheit berechnet werden kann und einen weiteren Rechenknoten anfordert. Es bleibt daher beinahe die halbe Abrechnungseinheit ungenutzt. Hier liegt noch Optimierungspotential bei der Länge der Zeitpauschale für das Herunterfahren des Rechenknotens. Die Abrechnungseinheiten der Cloud-Instanz werden in diesem Benchmark also im Durchschnitt nur zu rund zwei Drittel für Berechnungen genutzt. Auch wenn diese Zahl für andere Berechnungen höher sein kann, bleibt unweigerlich ein Overhead durch ungenutzte Teile von Abrechnungseinheiten.

Die Berechnung in der Cloud lieferte das Ergebnis also deutlich schneller, bei häufigem Durchführen von Berechnungen, beispielsweise in einem Unternehmen, können durch die Nutzung der Cloud-Ressourcen allerdings erhebliche Kosten entstehen.

4.2.1.1 Hybridsystem

Um sowohl das Problem des Auftretens von Lastspitzen, als auch das der vergleichsweise hohen Kosten für die Nutzung von Cloud-Ressourcen in den Griff zu bekommen bietet sich die Verwendung eines Hybridsystems aus beiden Ansätzen an. Dabei wird die Grundlast an Rechenkapazität vom Grid abgedeckt und zu Spitzenlastzeiten in die IaaS-Cloud hinaus skaliert. Die Cloud-Ressourcen werden also erst angefordert wenn alle Grid-Ressourcen belegt sind und noch weitere Berechnungsaufgaben vorliegen die noch nicht bearbeitet werden. Dieses Verhalten wird mittels Rechenknotengruppen mit unterschiedlichem zugeordnetem Kostenwert umgesetzt (siehe Abschnitt 3.1 *IaaS-Cloud-Anbindung*).

Die Berechnung des Anwendungsbeispiels mit unabhängigen Simulationen wurde auch auf einem Hybridsystem, das sowohl Grid-Computing-Anbindung als auch IaaS-Cloud-Anbindung verwendet, ausgeführt [24]. Dabei hatte das Grid-Computing-System wieder zehn Maschinen mit bis zu acht parallelen Berechnungen zur Verfügung. Im Sinne der Verwendung der Grid-Ressourcen zur Abdeckung der Grundlast, stellen diese zehn Maschinen den derzeit nicht belegten Teil eines größeren Grid-Computing-Systems dar. Die IaaS-Cloud-Anbindung verwendete abermals den heuristischen Ansatz mit Scheduling-Algorithmus zur Bestimmung der Menge an zu startenden Rechenknoten.

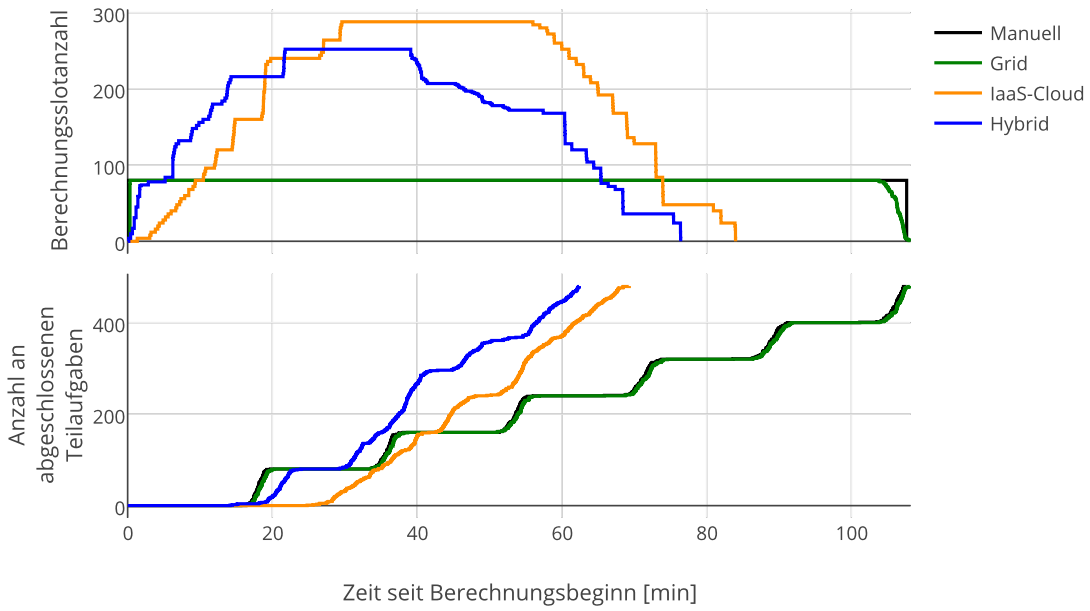

ABBILDUNG 4.13: ***Verlauf von Ressourcenmenge und Berechnungsfortschritt:*** *Das obere Diagramm stellt die Menge der über den Berechnungsverlauf hinweg zur Verfügung stehenden Berechnungsslots dar. Für die Berechnung in der Cloud steigt die Hardwaremenge anfangs langsamer als am Grid, da die Cloud-Instanzen Zeit zum Hochfahren benötigen und diese verzögert angefordert werden. Cloud- und Hybridlösung können aber mehr Berechnungsslots parallel zur Verfügung stellen. Das untere Diagramm zeigt die Fertigstellung der Teilaufgaben. Manuelle und Grid-Lösung liefern früher erste Ergebnisse da die ersten Teilberechnungen dort früher gestartet werden können und die Rechenleistung der Grid-Maschinen höher ist, fallen aber bald hinter Cloud- und Hybridlösung zurück da diese mehr Teilberechnungen parallel abarbeiten.*

Das Ergebnis der Berechnung des Transistors auf dem Hybridsystem (siehe Tabelle 4.1) zeigt, dass die Zeit bis zum Vorliegen des Gesamtberechnungsergebnis auf 58% des Wertes bei Berechnung nur auf dem Grid-Computing-System reduziert werden kann. Im Vergleich zur Berechnung unter Verwendung der IaaS-Cloud verkürzt sich die Berechnungsdauer zwar nur um 10%, die Anzahl der verbrauchten, vergleichsweise teuren Cloud-Instanz-Abrechnungseinheiten reduziert sich allerdings um mehr als 40%.

Für dieses Bespiel liefert das Hybridsystem also schneller Ergebnisse als die Nutzung von Grid-Computing-System oder Cloud-Computing alleine und die Kosten sind geringer als bei ausschließlicher Nutzung von Cloud-Ressourcen.

Für den Arbeitsablauf beim Simulieren von Bauelementen ist nicht nur relevant wann das Ergebnis der Berechnung vorliegt, sondern auch wann ein Trend in den Teilergebnissen erkennbar ist bzw. wann mit einer Vorauswertung begonnen werden kann. Mit Hilfe des Trends kann erkannt werden ob die Richtigen Parameter für die Simulation gewählt wurden, ober diese abgebrochen und mit neuen Parametern neu gestartet werden muss. Die Vorauswertung von Teilergebnissen kann die Zeit für eine Iteration des iterativen Arbeitsprozesses verkürzen.

Als Kennwerte für das Erkennen eines Trends bzw. Beginn der Vorauswertung sind in Tabelle 4.1 die Zeitpunkte, an denen 10% bzw. 90% der Teilberechnungen abgeschlossen sind, angeführt. Erwartungsgemäß wird die 10% Marke bei der Berechnung mit vorab manuell gestarteter Berechnungsknotensoftware etwas früher erreicht, als bei der Verwendung der Grid-Anbindung, da das Initialisieren der Berechnungsknotensoftware wegfällt. Bei Verwendung der IaaS-Cloud-Anbindung wird die 10% Marke erst erheblich später erreicht, da die Cloud-Instanzen mehr Zeit zum hochfahren benötigen und diese verzögert angefordert werden (siehe Abbildung 4.13). Die Berechnung auf dem Hybridsystem benötigt etwas mehr Zeit zum Erreichen der 10% Marke als die Berechnung bei Verwendung der Grid-Computing-Anbindung, da der zentrale Koordinationsserver hier auf einer Cloud-Instanz und nicht im lokalen Netzwerk des Grid-Computing-Systems ausgeführt wird. Aufgrund der geringeren Bandbreite und längeren Signalübertragungszeit verlängert sich daher die Zeitspanne, die zum Verteilen der Teilaufgaben benötigt wird.

Die 90% Marke wird bei Verwendung der IaaS-Cloud-Anbindung früher erreicht als bei Verwendung der Grid-Computing-Anbindung, da hier keine Limitierung der verfügbaren Hardware vorliegt und somit mehr Teilaufgaben parallel berechnet werden können. Die Hybridlösung kann schnell Teilberechnungen am Grid-Computing-System starten und dadurch früh Eingabedaten für den heuristischen Ansatz zur Bestimmung der Menge an zu startenden Rechenknoten sammeln. Außerdem können in der IaaS-Cloud weitere Instanzen gestartet werden um die Parallelität der Ausführung der Teilberechnungen zu maximieren. Daher erreicht die Hybridlösung das Maximum an verwendeter Ressourcen früher als die alleinige Nutzung der IaaS-Cloud-Anbindung (siehe Abbildung 4.13) und schließt die Berechnung von 90% der Teilaufgaben früher ab.

4.2.2 Simulationsketten zur Analyse von Nano-Bauelementen

Ein weiteres Anwendungsbeispiel, bei dem eine große Anzahl an Simulationen benötigt wird, ist die Analyse des “bias temperature instability (BTI)”-Effekts eines PMOS-Transistors. Dabei

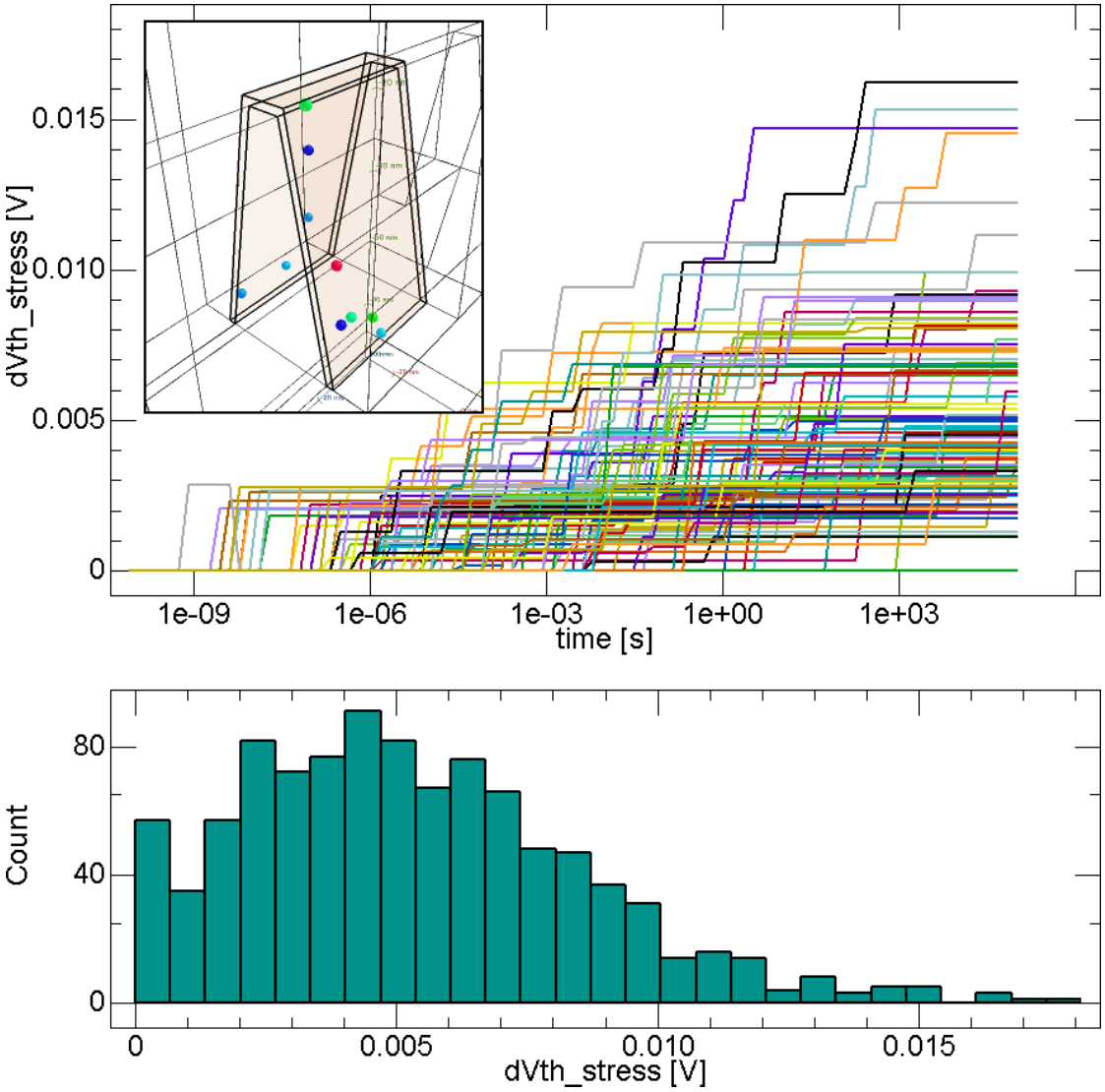

ABBILDUNG 4.14: *Zeitlicher Verlauf des durch BTI hervorgerufenen V_{th}-shift in einer Anzahl an n-FinFETs mit zufälligen Positionen und Eigenschaften der Oxid-Traps (oben); Verteilung der ΔV_{th} nach einer Stresszeit von 1×10^5 s (unten) (Quelle: [24])*

wird der Effekt analysiert, den geladene Traps im Oxid auf die Schwellenspannung V_{th} haben. Da die geladenen Traps lokal den gleichen Effekt haben wie eine am Gate anliegende Spannung, wird durch diese der Stromfluss zwischen Source und Drain beeinflusst. Damit ändert sich die Schwellenspannung V_{th} und damit die Charakteristik des Bauteils. Der Einfluss der Traps ist dabei von ihren Positionen und Eigenschaften abhängig. Daher ist eine große Anzahl an Simulationen nötig, bei denen die Traps zufällige Positionen und Eigenschaften haben [23].

Um die Analyse einer Trap-Konfiguration vorzunehmen sind drei Berechnungsschritte nötig. Im ersten Schritt werden deterministisch zufällige Werte für Postionen und Eigenschaften der Traps gewählt. Im zweiten Berechnungsschritt wird der Transistor unter "Stress" gesetzt indem am Gate eine negative Spannung angelegt wird. Nun laden sich die Traps und die zeitliche Änderung der Schwellenspannung V_{th} kann beobachtet werden (siehe Abbildung 4.14). Im dritten Schritt wird untersucht wie lange es dauert bis sich die Traps wieder entladen, wenn sich das Bauteil "entspannen" kann.

Zur Analyse einer Bauteilkonfiguration ist bei diesem Beispiel also eine Simulationskette bestehend aus drei voneinander abhängigen Einzelberechnungen nötig. Dabei werden die Berechnungsergebnisse der ersten Einzelberechnung als Eingabedaten für die zweite Berechnung benötigt und die der zweiten für die dritte. Der zentrale Koordinationsserver ist dabei auch für die Dateiübertragung zuständig. Diese ist so umgesetzt, dass Eingabedaten jeweils nach der Zuweisung einer Einzelberechnung zu einer Maschine zu dieser übertragen werden und danach die Berechnung gestartet wird. Die Berechnungsergebnisse werden zum Zentralen Koordinationsserver übertragen. Während diese Umsetzung sehr einfach nachzuvollziehen ist, nutzt sie viele Möglichkeiten der Optimierung nicht. So werden die Eingabedaten immer vom Koordinationsserver zum Rechenknoten übertragen, selbst wenn auf diesem Rechenknoten auch die Vorgängerberechnung durchgeführt wurde, die Daten also unter anderem Namen schon auf der Maschine vorhanden sind. Dieser Overhead könnte im Fall von voneinander abhängigen Berechnungen auf der selben Maschine oder auf in der Netzwerk-Topologie benachbarten Maschinen stark verringert werden und führt in der untersuchten Implementierung bei großen Dateien zu einer Überlastung des zentralen Koordinationsservers.

In diesem Beispiel werden dadurch für die drei Einzelberechnungen jeder Simulationskette in Summe 19.0 MB zu den Rechenknoten gesendet und 26.1 MB an Ergebnissen empfangen. Durch das Abarbeiten der Simulationsketten auf einer einzigen Maschine und durch das direkte Weiterverwenden der Ergebnisse könnten die übertragenen Eingabedaten von 19.0 MB auf 1.4 MB reduziert werden. Damit würde das Starten von Einzelberechnungen auf Rechenknoten deutlich beschleunigt. Da ein solcher Mechanismus bisher allerdings nicht implementiert ist, wird das Starten von Einzelberechnungen durch die Zeit, die für Dateiübertragungen benötigt wird, verzögert und das Grid-Computing-System kann nicht über die gesamte Berechnungsdauer voll ausgelastet werden (siehe Abbildung 4.15).

Noch deutlicher ist die Überlastung des zentralen Koordinationsservers bei der Simulation in der IaaS-Cloud zu sehen. Hier können die angeforderten Ressourcen, im Gegensatz zum Ablauf am Grid-Computing-System, in den ersten Minuten vollständig ausgelastet werden, da die Hardwaremenge nicht sprunghaft steigt. Die Gesamtmenge der zur Verfügung stehenden Hardware wird im Laufe der Berechnung allerdings deutlich höher und die Leistungsgrenze des zentralen

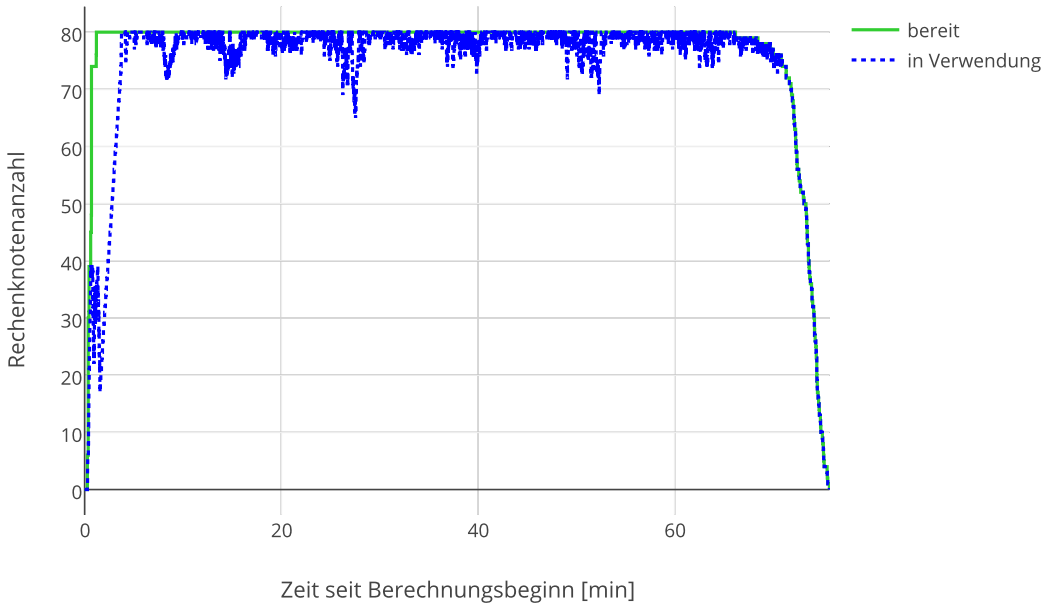

ABBILDUNG 4.15: ***Ablauf der Berechnung der Simulationsketten mit der Grid-Anbindung:*** *Zeitlicher Ablauf der Berechnung von 480 Simulationsketten bestehend aus je drei Einzelberechnungen auf einem Grid-Computing-System mit zehn Maschinen mit je acht parallelen Berechnungen. Dabei wird zu Beginn in kurzer Zeit für 39 Simulationsketten jeweils die erste Einzelberechnung gestartet. Diese hat nur 1.4 MB an Eingabedaten und eine Laufzeit von etwa 24 Sekunden. Sobald Ergebnisse des ersten Berechnungsschritts einer Simulationskette vorliegen, wird bevorzugt der zweite Berechnungsschritt dieser Kette gestartet anstatt weitere erste Berechnungsschritte zu starten um der Benutzerin oder dem Benutzer möglichst früh Ergebnisse einer abgeschlossenen Simulationskette präsentieren zu können. Der zweite Berechnungsschritt benötigt allerdings 8.8 MB an Eingabedaten. Durch die Zeit, die für die Datenübertragung benötigt wird, verzögert sich das Starten der Einzelberechnungen und das Grid-Computing-System kann erst nach 3.75 Minuten erstmals vollständig ausgelastet werden. Auch im weiteren Verlauf sinkt die Auslastung immer wieder wenn mehre Einzelaufgaben in kurzer Zeit fertig werden.*

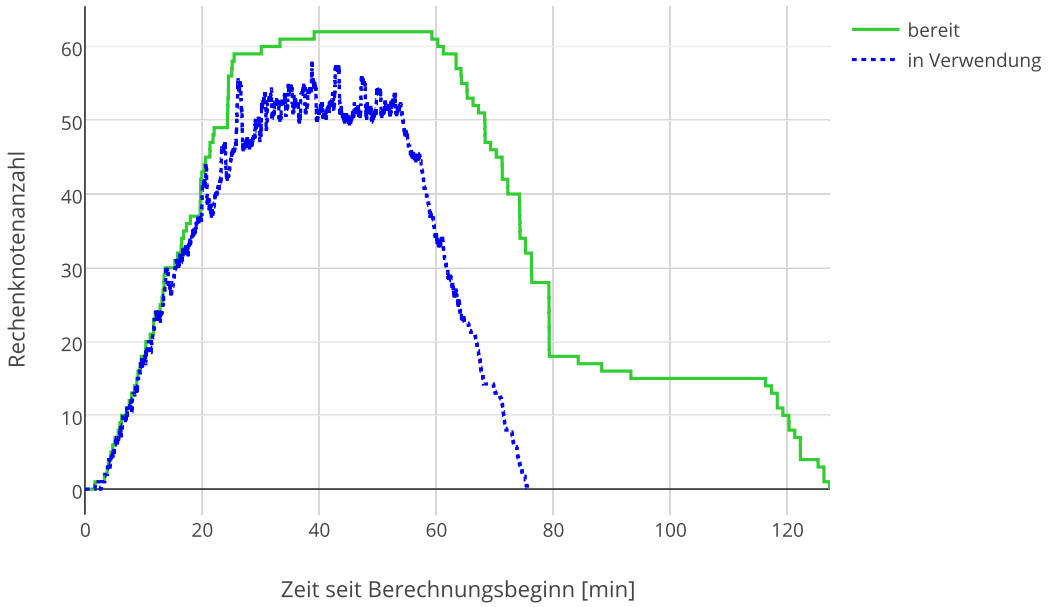

ABBILDUNG 4.16: ***Ablauf der Berechnung der Simulationsketten mit der Cloud-Anbindung:*** *Zeitlicher Ablauf der Berechnung von 480 Simulationsketten bestehend aus je drei Einzelberechnungen in der IaaS-Cloud. Jeder Rechenknoten kann bis zu vier Einzelberechnungen parallel bearbeiten. Durch die Verwendung des heuristischen Ansatz mit Scheduling-Algorithmus zur Bestimmung der Menge an zu startenden Rechenknoten werden die Rechenknoten nach und nach angefordert. Daher können diese in den ersten 20 Minuten noch vollständig ausgelastet werden. Danach benötigt die serielle Dateiübertragung vom zentrale Koordinationsserver zu den Rechenknoten länger als die parallele Abarbeitung der Einzelberechnungen und die angeforderte Hardware kann daher nicht vollständig ausgelastet werden.*

Koordinationsservers damit überschritten (siehe Abbildung 4.16). Dadurch kommt der Vorteil der massiv parallelen Berechnung bei der Verwendung der IaaS-Cloud nicht mehr zu tragen und damit wird die Berechnung nicht signifikant früher abgeschlossen (siehe Tabelle 4.2 und Abbildung 4.17). Außerdem entstehen durch die ungenutzten angeforderten Ressourcen zusätzliche Kosten. Diese könnten vermieden werden indem der zentralen Koordinationsserver mit einem Limit an Berechnungskonten konfiguriert wird, dazu muss die Belastungsgrenze allerdings bekannt sein. Diese Belastungsgrenze ist jedoch vom Verhältnis von Datenübertragung zu Berechnungszeit abhängig, variiert also je nach Gesamtberechnung.

Für dieses Anwendungsbeispiel bringt die Berechnung in der IaaS-Cloud ohne die oben genannten Verbesserungen gegenüber der Berechnung auf dem Grid-Computing-System also keine Vorteile bei der Berechnungszeit, jedoch höhere Kosten da weniger leistungsfähige Prozessoren und langsamere Netzwerkverbindungen durch eine höhere Anzahl an Rechenknoten

	10% fertig [min]	90% fertig [min]	Berechnungs-dauer [min]	Kosten
Manuell	13.9	73.6	75.3	12.5 Maschinenstunden (8 Berechnungsslots pro Maschine)
Grid	14.3	73.4	75.8	12.1 Grid-Maschinenstunden (8 Berechnungsslots pro Maschine)
IaaS-Cloud	31.0	73.8	75.6	77 Cloud-Instanz-Abrechnungseinheiten (4 Berechnungsslots pro Instanz)
Hybrid (extrapoliert)	636.2	1171.2	1228.8	über 300 Grid-Maschinenstunden über 800 Cloud-Abrechnungseinheiten

TABELLE 4.2: ***Berechnungsdauern und Kosten im Beispiel mit Simulationsketten:*** *Der Overhead der Grid-Anbindung gegenüber dem manuellen Starten der Rechenknoten fällt mit einer halben Minute kaum ins Gewicht. Die Berechnung in der IaaS-Cloud kann das Gesamtergebnis nicht signifikant schneller liefern, da die Teilaufgaben Aufgrund der Überlastung des zentralen Koordinationsservers weniger stark parallelisiert werden als möglich.*

ausgeglichen werden müssen um in die Ergebnisse in der gleichen Zeit zu erhalten. Das Kriterium, das bei diesem Anwendungsbeispiel für die schlechte Eignung für die Berechnung in der IaaS-Cloud verantwortlich ist, ist das Verhältnis von Eingabedatengröße zu Berechnungszeit. Auch die Ausgabedatengröße und die Berechnungsdauer von Einzelberechnungen haben hier Einfluss.

Bei der Berechnung des Anwendungsbeispiels mit Hilfe eines Hybridsystems fällt das Problem der Übertragung von Eingabedaten vom zentralen Koordinationsserver zu den Rechenknoten gegenüber Grid-Computing-System und IaaS-Cloud noch deutlich mehr ins Gewicht. Während sich bei der Verwendung von Grid-Computing-System bzw. IaaS-Cloud der zentrale Koordinationsserver und alle Rechenknoten am selben Standort befinden, ist dies bei Verwendung des Hybridsystems nicht der Fall. Daher ist die verfügbare Bandbreite zu einigen der Rechenknoten deutlich geringer. Um die Vergleichbarkeit der Messergebnisse zu gewährleisten wurde wie in Unterabschnitt 4.2.1 (*Unabhängige Simulationen von Nano-Bauelementen*) der zentrale Koordinationsserver in der IaaS-Cloud betrieben.

Der zentrale Koordinationsserver ist so konfiguriert, dass zuerst das Grid-Computing-System voll ausgelastet wird und erst danach Berechnungen in der IaaS-Cloud gestartet werden. Die Menge an angeforderter Hardware in der IaaS-Cloud richtet sich nach der erwarteten benötigten Gesamtmaschinenzeit, ohne Berücksichtigung der tatsächlichen Auslastung der bereits angeforderten Maschinen. Daher werden IaaS-Cloud Ressourcen angefordert obwohl diese nie ausgelastet werden können, da die Übertragung der Eingabedaten zu den Rechenknoten des Grid-Computing-Systems über eine Verbindung mit geringer Bandbreite den zentrale Koordinationsserver beinahe zum Stillstand bringt. Das Ergebnis einer Berechnung mit Eingabedaten dieser Größe auf einem Hybridsystem wäre also, dass Berechnungsdauer und Kosten jene auf einem Grid-Computing-System um Größenordnungen übersteigen (siehe Tabelle 4.2).

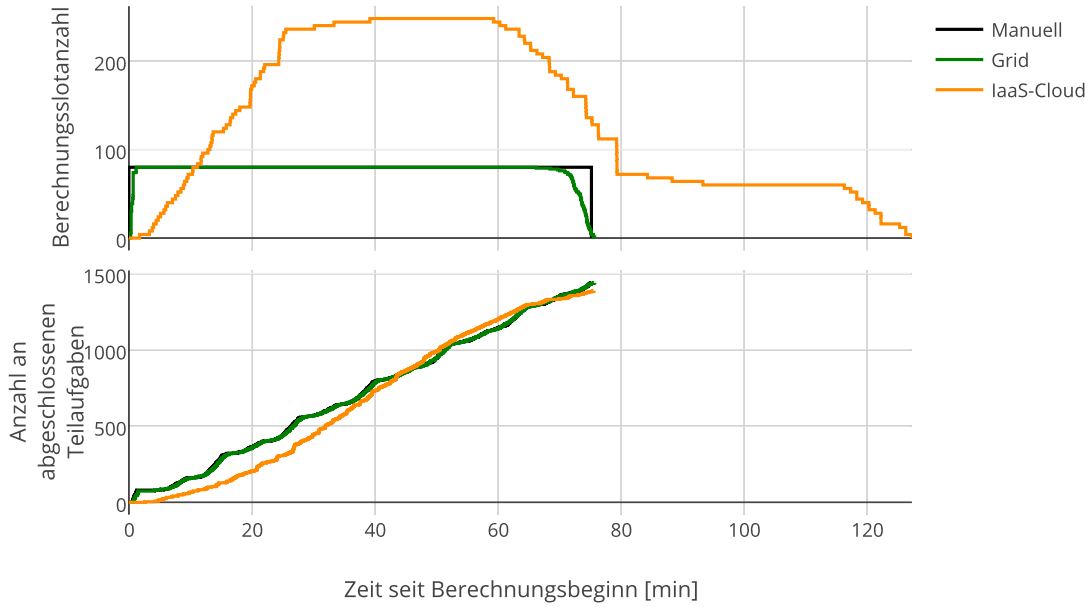

ABBILDUNG 4.17: ***Verlauf von Ressourcenmenge und Berechnungsfortschritt:*** *Der Vergleich der unterschiedlichen Hardware-Anbindungen zeigt, dass die Berechnung in der IaaS-Cloud bei dieser Anwendung trotz deutlich höherer Hardwaremenge die Berechnungsergebnisse nicht schneller liefern kann. Das kann dadurch erklärt werden, dass der vom zentralen Koordinationsserver ausgehende Dateitransfer die Abarbeitungsgeschwindigkeit limitiert hat, nicht die Rechenkapazität.*

Die dafür verantwortlichen Dateitransfer-Probleme können aber durch eine Reihe von Maßnahmen verringert werden. So kann die übertragene Datenmenge deutlich verringert werden, indem voneinander abhängige Teilaufgaben auf derselben Maschine berechnet werden und somit die Ausgabedaten der ersten Teilaufgabe direkt als Eingabedaten der zweiten Teilaufgabe weiterverwendet werden können. Außerdem kann die Bandbreite für Übertragungen erhöht werden, indem die Daten verteilt gespeichert werden. Dies kann beispielsweise durch mehrere Dateiserver, mindestens einen pro Berechnungsknotenstandort, durch ein verteiltes Dateisystem wie Gluster [25] oder durch peer-to-peer-Dateiübertragung zwischen den Rechenknoten umgesetzt werden.

KAPITEL 5

Qualitativer Vergleich mit verwandten Arbeiten

Die Idee ein Batch-Queuing-System um die Fähigkeit der Skalierbarkeit zu erweitern, wurde auch in [26] behandelt und in [27] und [4] weiterverfolgt. Der in jener Arbeit vorgestellte Ansatz erweitert das Nimbus toolkit [28] um die Fähigkeit Rechencluster dynamisch mit Ressourcen einer privaten IaaS-Cloud zu erweitern. Außerdem werden unterschiedliche Verhaltensmuster für verschiedene Anforderungsprofile vorgeschlagen. Die zwei vorgestellten Extremfälle an Anforderungsprofilen sind “Steady stream”, bei dem Berechnungsaufgaben kontinuierlich in regelmäßigen Abständen eingereicht werden, und “Bursts”, bei dem eine große Menge an Berechnungen auf einmal ankommen. Die Benchmarks in dieser Arbeit sind vergleichbar dem “Bursts”-Profil, da dieses für Bauteilsimulationen typisch ist.

In [27] wird der Algorithmus zur Verwaltung der IaaS-Instanzen so verändert, dass Maschinen nur heruntergefahren werden, wenn sich diese im Leerlauf befinden und vor der nächsten Evaluierung eine neue Abrechnungseinheit verbrauchen. Ein ähnlicher Ansatz wurde auch in dieser Arbeit verfolgt. Allerdings kann die Dauer eines Durchlaufs des Instanz-Evaluierungs-Algorithmus bei unterschiedlicher Last oder Anzahl an verwalteten Maschinen schwanken. Daher wurde für diese Dauer und das Herunterfahren der Instanz selbst eine pauschalierte Zeitspanne von 5 Minuten verwendet.

Einer der Hauptunterschiede zu dieser Arbeit ist, dass in [26] und [27] nicht auf das stundenweise Abrechnungsmodell Rücksicht genommen wird, da der Fokus jener Arbeiten auf der Verwendung in einer private Cloud liegt. Außerdem werden in dieser Arbeit Hybridsysteme bestehend aus Grid-Computing-Systemen und IaaS-Clouds untersucht. Dabei zeigt sich, ein deutlich verändertes Verhalten im Vergleich zur Verwendung einer einzelnen Cloud.

Weiters wird in [27] ein Möglichkeit der Kostenkontrolle in Hybriden Berechnungssystemen vorgeschlagen. Dabei wird wird ein stündliches Budget für Cloud-Ressourcen festgelegt, beispielsweise 5 Dollar. Das Geld akkumuliert, sodass beispielsweise nachdem 3 Stunden lang keine Cloud-Ressourcen verwendet wurden 15 Dollar für Cloud-Ressourcen zur Verfügung stehen um Lastspitzen abzufangen. Dadurch können die Kosten kontrolliert werden, ohne die

maximalen parallel nutzbaren Ressourcen limitieren zu müssen. Dies ist erstrebenswert, da genau die hohe Anzahl der parallel nutzbaren Ressourcen einer der größten Vorteile der IaaS-Cloud-Nutzung ist. Außerdem stellt die Methode eine kontinuierliche Verfügbarkeit von Cloud-Ressourcen sicher. Anders als bei Limits für eine längere Zeitspanne (beispielsweise monatlich), bei denen das Budget innerhalb eines Bruchteils der Zeitspanne ausgeschöpft sein kann.

Der in [4] vorgestellte Ansatz basiert auf anderen Software-Lösungen als [26]. Er erweitert den open-source Manager-Service Elastic Processing Unit (EPU) [29] um einen zentralen Scheduler, der Rechenknoten überwacht und Berechnungsaufgaben auf diesen startet. Die Architektur entspricht also jener, die auch in dieser Arbeit gewählt wurde.

Außerdem wurde das Konfigurations- und Managementtool Chef [30] genutzt um manuelles Konfigurieren und Verwalten von Systemabbildern von Cloud-Instanzen zu vermeiden und somit unabhängig von IaaS-Cloud-Anbieten zu sein. Dieser Ansatz hat allerdings den Nachteil, dass die Zeit, welche vom Anfordern eines Rechenknotens bis zu dessen Einsatzbereitschaft vergeht, verlängert wird. Um die Benchmark-Ergebnisse nicht mit diesem vermeidbaren Overhead zu verfälschen, wurden die Benchmarks dieser Arbeit mit einem vorgefertigten Systemabbild durchgeführt. Für die Unterstützung einer großen Anzahl an IaaS-Cloud-Anbietern kann es aber wirtschaftlich sein den Overhead in Kauf zu nehmen.

Bei der Auswertung des Berechnungsablaufs in [4] wurde festgestellt, dass es möglich ist 476 Instanzen in weniger als 15 Minuten anzufordern. Auch bei den Benchmarks in dieser Arbeit wurden ähnlich hohe Werte erreicht. Während des Benchmarks in Unterabschnitt 4.2.1 (*Unabhängige Simulationen von Nano-Bauelementen*) werden bis zu 20 zusätzliche Instanzen auf einmal benötigt (siehe Abbildung 4.12). Diese werden durch 20 API-Aufrufe innerhalb von 34 Sekunden angefordert. 51 Sekunden nach dem ersten API-Aufruf ist der erste Rechenknoten zur Verwendung bereit und 74 Sekunden später kann auf allen Knoten gerechnet werden. Unter der Annahme, dass die API-Aufrufe der limitierende Faktor sind, könnten, hochgerechnet auf die in [4] betrachtet Zeitspanne von 15 Minuten, nach dieser Messung theoretisch über 500 Instanzen angefordert werden. Um diese Instanzen mit je vier oder mehr CPUs mit Berechnungsaufgaben und Eingabedateien aufzufüllen, würde allerdings weit mehr Zeit benötigt, so dass der limitierende Faktor bei der Implementierung in dieser Arbeit wohl in diesem Bereich zu suchen ist. Besonders bei Berechnungen mit großen Eingabedateien sinkt hier der Berechnungsdurchsatz.

Für die Anzahl der zu startenden Instanzen wird bei [4] direkt die Zahl der Teilaufgaben verwendet. Die Randbedingung der stundenweisen Abrechnung wird hier also nicht beachtet. Für Anwendungen mit einer hohen Anzahl an kurzen Teilberechnungen, wie sie im Bereich der Bauelemente-Simulation häufig vorkommen, ist die Kostenminimierung durch Laufzeitabschätzung (siehe Unterabschnitt 3.1.1 *Grundlegende Architektur*) allerdings unverzichtbar.

Weiters wird in [4] beobachtet, dass angeforderte Instanzen teilweise nicht erfolgreich hochfahren oder Probleme haben, sich mit dem Netzwerk zu verbinden. Diese Beobachtung wurde auch während der Benchmarks für diese Arbeit gemacht. Allerdings waren nur einzelne Instanzen in wenigen Benchmarks betroffen, während in [4] bei 22 von 512 Instanzen das Hochfahren fehlschlägt. Um dieses Problem in den Griff zu bekommen, wird in beiden Arbeiten nach einer Zeitüberschreitung eine Ersatzinstanz gestartet. Als vorsorgliche Maßnahme mehr Instanzen

anzufordern als benötigt, wie in in [4] vorgeschlagen, scheint laut den Erfahrungen in dieser Arbeit nicht oder nur bei einer sehr großen Anzahl an verwendeten Instanzen nötig.

Ein anderer Ansatz der Ressourcennutzung über ein Grid-Computing-System hinaus wird in [31] untersucht. Hier wird nicht nur die Nutzung von IaaS-Clouds sondern auch der Lastaustausch zwischen mehreren Grid-Computing-Systemen betrachtet. Die Algorithmen zur Aufteilung der Berechnungen auf unterschiedliche Grid-Computing-Systeme oder die IaaS-Cloud bzw. auf einzelne Rechenknoten werden dabei auch mittels maschinellem Lernen entwickelt. Allerdings wird das stundenweise Abrechnungsmodell von IaaS-Cloud-Betreibern auch hier nicht berücksichtigt.

KAPITEL 6

Weiterführende Forschung

Die Erweiterung um die autonome Skalierbarkeit des Batch-Queuing-Systems weitet dessen Einsatzmöglichkeiten aus. Bei der Erstellung der Benchmarks und im Praxiseinsatz wurden allerdings mögliche weitere Verbesserungsmöglichkeiten gefunden. Die Heuristik zur Bestimmung der Menge an zu startenden Rechenknoten führt Berechnungen basierend auf Informationen von bereits abgeschlossenen und noch laufenden Berechnungsteilaufgaben durch (siehe Unterabschnitt 3.1.1 *Grundlegende Architektur*). Sind noch keine solchen Informationen vorhanden, da bisher keine vergleichbaren Berechnungsaufgaben gestartet wurden, so nimmt die Heuristik eine sehr kurze Berechnungsdauer an, um nicht zu viele Ressourcen anzufordern. Das Starten von Ressourcen kann aber beschleunigt werden, indem in diesem Fall mit einer anwendungstypischen Mindestdauer gerechnet wird. Noch bessere Ergebnisse können vermutlich mit einem für jede Berechnung individuellen Wert erzielt werden, der aus vorhergehenden Berechnungen mit anderen Parametern stammt, oder aufgrund der örtlichen Berechnungsauflösung geschätzt wird. Diese Abschätzung ist zwar nicht genau genug um sie als Erwartungswert für die Heuristik zu verwenden, kann aber für die Berechnung einer zu erwartenden Mindestberechnungsdauer verwendete werden.

Das verwendete Batch-Queuing-System priorisiert Berechnungen ausschließlich nach ihrem Alter, sie werden also nach dem “First In – First Out”-Prinzip abgearbeitet. Verwenden jede Benutzerin und jeder Benutzer einen eigenen zentralen Koordinationsserver, der die Berechnungen unter dem eigenen BenutzerInnennamen an ein Grid-Computing-System weiterreicht, das faire Verteilung der Ressourcen und Priorisierung unterstützt, so ist “FIFO”-Ordnung ausreichend. Wird jedoch ein Koordinationsserver von mehreren Benutzerinnen und Benutzern verwendet, so fehlen diese Funktionalitäten. Ein weiterer Vorteil der interen Bestimmung der Abarbeitungsordnung wäre, dass die schon vorhandenen Laufzeitabschätzungen mitbenutzt werden können, um kurze Berechnungen automatisch zu Priorisieren.

Außerdem besitzt das verwendete Batch-Queuing-System zwar die Möglichkeit einen Simulator auf mehreren Prozessoren gleichzeitig auszuführen indem mehrere Berechnungsslots auf

dieser Maschine belegt werden, einem Simulator Berechnungsslots auf unterschiedlichen Maschinen zuzuweisen ist jedoch nicht möglich. Für Simulatoren die selbst die Fähigkeit besitzen verteilt zu arbeiten, beispielsweise mittels MPI, würde dies die Parallelisierbarkeit der Berechnungen weiter erhöhen.

Besonders bei Berechnungen mit großen Eingabedateien sinkt der Berechnungsdurchsatz der in dieser Arbeit vorgestellten Implementierung erheblich, da der zentrale Koordinationsserver auch für die Dateiübertragung zuständig ist und sich damit die hohe Belastung durch den Datentransfer auch negativ auf dessen andere Aufgaben auswirken. Eine mögliche Lösung beseht darin, alle Aufgaben die den Dateitransfer betreffen auszulagern, eventuell sogar auf eine andere Maschine, und deren Effizienz zu optimieren. In [4] werden gute Resultate mit der Verwendung des Gluster Dateisystems [25] für diesen Zweck beschrieben. Außerdem können Dateiübertragungen vermieden werden, indem voneinander abhängige Teilaufgaben auf derselben Maschine berechnet werden und somit die Ausgabedaten der ersten Teilaufgabe direkt als Eingabedaten der zweiten Teilaufgabe weiterverwendet werden können, ohne diese über das Netzwerk transportieren zu müssen.

Eine weitere Verbesserungsmöglichkeit besteht darin, das Fehlschlagen von Teilberechnungen aufgrund von Fehlern der Netzwerkverbindung, der Maschine oder der Simulation zu erkennen und diese auf einer anderen Maschine neu zu starten. Derzeit werden solche Teilberechnungen in der graphischen BenutzerInnenoberfläche als fehlgeschlagen markiert. Ein erneuter Berechnungsversuch muss aber manuell gestartet werden.

KAPITEL 7

Zusammenfassung und Fazit

Im Rahmen der vorliegenden Arbeit wurde ein bestehendes Batch-Queuing-System [3] um jeweils eine Anbindung zu einem Grid-Computing-System und einer IaaS-Cloud erweitert. Der Performance-Overhead dieser zusätzlichen Abstraktionsschicht fällt bei den typischen Berechnungsdauern im Einsatzgebiet Nano-TCAD nicht ins Gewicht. Die Grid-Computing-Anbindung bietet also bei vergleichbarer Performance zum manuellen Starten von Rechenknoten den Vorteil bestehende Grid-Infrastruktur mitbenutzen zu können und die gemeinsame Nutzung dieser Ressourcen mit anderen Applikationen verwalten zu können.

Für die IaaS-Cloud Anbindung ist es unter Berücksichtigung der stundenweisen Abrechnung und der Hochlaufzeit der Rechenknoten aus Kostengründen nicht ideal alle Teilaufgaben einer Berechnung parallel zu berechnen. Daher wurde eine Heuristik zur Bestimmung der Menge an zu startenden Rechenknoten entwickelt, deren Ziel es ist bei minimalen Kosten die Parallelität der Berechnungen zu maximieren. Bei Verwendung dieser Heuristik wird aus Kostengründen nicht die kürzest mögliche Wartezeit auf das Gesamtergebnis angestrebt, dennoch kann diese im Vergleich zur Berechnung auf einem Grid-Computing-System deutlich reduziert werden, insbesondere für besonders lange Berechnungsdauern.

Ein Hybridsystem, das sowohl Grid-Computing-System als auch IaaS-Cloud nutzt, kann schneller Ergebnisse liefern als die Nutzung von Grid-Computing-System oder Cloud-Computing alleine und dabei geringere Kosten verursachen als die ausschließlicher Nutzung von Cloud-Ressourcen. Diese Vorteile kommen allerdings nur bei einer geringen Größe der Eingabedaten zu Tragen, da sonst die für Datenübertragung benötigte Zeit die Vorteile überwiegt. Das in dieser Arbeit analysierte Hybridsystem ist bereits für viele zeitkritische, rechenintensive Anwendungen im Nano-TCAD praktikabel und ökonomisch. Außerdem wurden Möglichkeiten aufgezeigt, den Datentransfer effizienter und skalierbarer umzusetzen.

Es wurde gezeigt, dass ein autonom skalierendes Berechnungsnetzwerk durch den kurzfristigen Zugriff auf mehr Ressourcen schneller Berechnungsergebnisse liefern kann als ein statisches und dabei die Kosten minimal gehalten werden können. Außerdem wurde ein Hybridsystem entwickelt, das sowohl Berechnungszeit als auch Kosten gegenüber Cloud-Computing verringern kann.

Literaturverzeichnis

[1] *Sun Grid Engine 5.2.3 Manual*, Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, CA 94303-4900 U.S.A., July 2001.

[2] *Platform LSF*. http://ibm.com/systems/platformcomputing/products/lsf/

[3] P. Prause, "Design and Integration of Distributed Computing Concepts in a TCAD Framework," Master's thesis, Technische Universität Wien, September 2008.

[4] P. Marshall, H. Tufo, K. Keahey, D. LaBissoniere, and M. Woitaszek, "A Large-Scale Elastic Environment for Scientific Computing," in *Software and Data Technologies*, ser. Communications in Computer and Information Science, J. Cordeiro, S. Hammoudi, and M. van Sinderen, Eds. Springer Berlin Heidelberg, 2013, vol. 411, pp. 112–126.

[5] J. Tang, W. P. Tay, and Y. Wen, "Dynamic Request Redirection and Elastic Service Scaling in Cloud-Centric Media Networks," *Multimedia, IEEE Transactions on*, vol. 16, no. 5, pp. 1434–1445, Aug 2014.

[6] Michael Armbrust, Armando Fox, R. Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, G. Lee, D. A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing," Feb 2009.

[7] M. Mao, J. Li, and M. Humphrey, "Cloud auto-scaling with deadline and budget constraints," in *Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on*, Oct 2010, pp. 41–48.

[8] P. Saripalli and B. Walters, "Quirc: A quantitative impact and risk assessment framework for cloud security," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*. Ieee, 2010, pp. 280–288.

[9] B. Jaspan, "Improved Persistent Login Cookie Best Practice," http://jaspan.com/improved_persistent_login_cookie_best_practice, Abgerufen: 05.07.2016.

[10] F. Rocha and M. Correia, "Lucy in the sky without diamonds: Stealing confidential data in the cloud," in *Dependable Systems and Networks Workshops (DSN-W), 2011 IEEE/IFIP 41st International Conference on*, June 2011, pp. 129–134.

[11] C. Gentry, “A fully homomorphic encryption scheme,” Ph.D. dissertation, Stanford University, 2009.

[12] M. Naehrig, K. Lauter, and V. Vaikuntanathan, “Can homomorphic encryption be practical?” in *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*. ACM, 2011, pp. 113–124.

[13] M. Balduzzi, J. Zaddach, D. Balzarotti, E. Kirda, and S. Loureiro, “A security analysis of amazon’s elastic compute cloud service,” in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, ser. SAC ’12. New York, NY, USA: ACM, 2012, pp. 1427–1434. http://doi.acm.org/10.1145/2245276.2232005

[14] *Amazon Elastic Compute Cloud*, Amazon Web Services, April 2015. http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-ug.pdf

[15] *Metrics User Manual*. https://dropwizard.github.io/metrics/3.0.2/manual/

[16] *Ganglia*. http://sourceforge.net/projects/ganglia/

[17] *Graphite*. http://graphite.wikidot.com/documentation

[18] “GTS Framework.” http://www.globaltcad.com/framework

[19] H. Demel, “Verteiltes Simulationsframework,” 2011. https://dl.dropboxusercontent.com/u/353069/forever/demel_0728129_bachelorarbeit.pdf

[20] N. Sano and M. Tomizawa, “Random dopant model for three-dimensional drift-diffusion simulations in metal–oxide–semiconductor field-effect-transistors,” *Applied Physics Letters*, vol. 79, no. 14, pp. 2267–2269, 2001. http://scitation.aip.org/content/aip/journal/apl/79/14/10.1063/1.1406980

[21] H.-W. Cheng, F.-H. Li, M.-H. Han, C.-Y. Yiu, C.-H. Yu, K.-F. Lee, and Y. Li, “3D device simulation of work function and interface trap fluctuations on high-K / metal gate devices,” in *Electron Devices Meeting (IEDM), 2010 IEEE International*, Dec 2010, pp. 15.6.1–15.6.4.

[22] A. Asenov, S. Kaya, and A. Brown, “Intrinsic parameter fluctuations in decananometer MOSFETs introduced by gate line edge roughness,” *Electron Devices, IEEE Transactions on*, vol. 50, no. 5, pp. 1254–1260, May 2003.

[23] T. Grasser, B. Kaczer, W. Goes, T. Aichinger, P. Hehenberger, and M. Nelhiebel, “A two-stage model for negative bias temperature instability,” in *Reliability Physics Symposium, 2009 IEEE International*. IEEE, 2009, pp. 33–44.

[24] H. Demel, Z. Stanojevic, M. Karner, G. Rzepa, and T. Grasser, “Expanding TCAD Simulations from Grid to Cloud,” in *2015 International Conference on Simulation of Semiconductor Processes and Devices (SISPAD)*. IEEE, Sept 2015, pp. 186–189.

[25] *Gluster*. http://www.gluster.org

[26] P. Marshall, K. Keahey, and T. Freeman, "Elastic Site: Using Clouds to Elastically Extend Site Resources," in *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, ser. CCGRID '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 43–52. http://dx.doi.org/10.1109/CCGRID.2010.80

[27] P. Marshall, H. Tufo, and K. Keahey, "Provisioning policies for elastic computing environments," in *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*. IEEE, 2012, pp. 1085–1094.

[28] W. Huang, J. Liu, B. Abali, and D. K. Panda, "A case for high performance computing with virtual machines," in *Proceedings of the 20th annual international conference on Supercomputing*. ACM, 2006, pp. 125–134.

[29] K. Keahey, P. Armstrong, J. Bresnahan, D. LaBissoniere, and P. Riteau, "Infrastructure Outsourcing in Multi-cloud Environment," in *Proceedings of the 2012 Workshop on Cloud Services, Federation, and the 8th Open Cirrus Summit*, ser. FederatedClouds '12. New York, NY, USA: ACM, 2012, pp. 33–38. http://doi.acm.org/10.1145/2378975.2378984

[30] A. Jacob, "Infrastructure in the Cloud Era," in *International OReilly Conference Velocity*, 2009.

[31] A. Fölling, "Effiziente Kapazitätsplanung durch dynamische Erweiterung einer lokalen Ressourcenumgebung um Grid-und Cloud-Ressourcen: algorithmische und technische Betrachtungen," Ph.D. dissertation, 2014.