7. März 2019

Parallele Dateisysteme

Autor: Phillipp Jäger
Betreuer: Eugen Betke

Modul: Speicher- und Dateisysteme
Leitung: Michael Kuhn

Inhaltsverzeichnis

1. Einleitung

1.1 Motivation

Big Data sowie ein großer Teil der Wissenschaft haben eine herausstechende Gemeinsamkeit: eine große Menge an Daten. Die schier unendliche Menge an Daten, die tagtäglich analysiert, verarbeitet und gespeichert werden müssen stellt die Akteure vor eine große Herausforderung. So genannte *Flaschenhalse* (engl. *bottlenecks*) sind die Stellen eines Systems, die als limitierender Faktor für die Leistung bzw. den Output des gesamten Systems verantwortlich sind.[1] Eine Verbesserung des Flaschenhalses ist meist direkt am Endergebnis messbar, wohingegen die Verbesserung anderer Faktoren so gut wie gar keine Auswirkung hat. In dem Bereich des *High Performance Computing (HPC)*, wo oft tausende bis Millionen von Prozessen gleichzeitig auf die Speichermedien zugreifen, wurde zunehmend der Prozess der *Ein- und Ausgabe* (engl. *Input-Output*, auch: *I/O*) zum Flaschenhals.[2]

Doch nicht nur die Datendurchsatzrate ist ein limitierender Faktor bei der Speicherung, auch die Kapazität ist ein limitierender Faktor für ein *HPC*-System.[3] Hier liegt der Fokus der Konstruktion paralleler Dateisysteme ebenfalls auf dem Verhindern von Flaschenhalsen und somit der optimalen Nutzung der zur Verfügung stehenden Speicherkapazität.

1.2 Definition

Ein paralleles Dateisystem ist zunächst wie jedes Dateisystem als *Middleware* zwischen Betriebssystem und Laufwerk(en) zu verstehen. Als solches regelt es die Benennung, Verteilung, *persistente Speicherung* (Speicherung über lange Zeit hinweg) und Organisation von Daten auf den Speichermedien. Für die Organisation wird ein *Wurzelverzeichnis* (engl. *root directory*) genutzt, in dem Dateien oder weitere Verzeichnisse liegen können, sodass sich eine *Baumstruktur* ergibt. Innerhalb einer Ebene gibt es einen festen *Namensraum* (engl. *namespace*), der die Benennung der Dateien festlegt und dafür sorgt, dass jedes Objekt eindeutig anhand seines *Pfadnamens* identifiziert werden kann. Die Dateien besitzen sogenannte *Metadaten*, welche genauere Informationen über verschiedene Aspekte der Datei beinhalten. Beispiele für solche Informationen sind der Autor, Zugriffsrechte und das Erstelldatum der Datei.

[1] https://martinvogel.de/lexikon/bottleneck.html [entnommen am 27.02.2019]
[2] Markus Ermes: Verteilte, parallele Dateisysteme, 18.12.2018
[3] http://www.linux-magazin.de/ausgaben/2007/11/need-for-speed/ [entnommen am 28.02.2019]

Diese Fähigkeiten eines Dateisystems kann ein paralleles Dateisystem, wie der Name impliziert, parallel erfüllen. Das heißt es wird ein paralleler Zugriff auf die im Dateisystem eingebundenen Ressourcen ermöglicht und dadurch erreicht wird, dass die Rechenknoten eine möglichst hohe Auslastung haben und der Dateizugriff sowie die Dateispeicherung möglichst effizient erfolgen.

1.3 Abgrenzung zu anderen Dateisystemen

Ein lokales Dateisystem, ist für den Gebrauch von einem Nutzer an einem Rechner ausgelegt ist und somit nur den physisch direkt angeschlossenen Speicher verwaltet. Auch ein lokales Dateisystem erlaubt den parallelen Zugriff auf Daten, jedoch mit einem Zeitaufwand, der in Hochleistungsrechnern zum *Flaschenhals* werden würde. Des Weiteren würde der Einsatz von mehreren lokalen Dateisystemen in einem Netzwerk dazu führen, dass jedes System seinen eigenen Namensraum verwenden würde und somit Dateien und Verzeichnisse mit identischen Bezeichnungen auf den verschiedenen Knoten existieren könnten.

Die Abgrenzung zu verteilten Dateisystemen hingegen ist schwierig, denn parallele Dateisysteme sind eine Unterart von verteilten Dateisystemen und sind somit sehr ähnlich wie verteilten Dateisysteme aufgebaut.[4] Hauptkennzeichen der verteilten Dateisysteme ist die Entkopplung und Verteilung von Speicherung und Berechnung. *Rechenknoten* (engl. *compute nodes*) sind für das Ausführen von vielen Operationen dedizierte Einheiten, die über ein Netzwerk mit den *Speicherknoten* (engl. *storage nodes*) verbunden sind. Diese Komponenten können daher im gleichen Raum oder Gebäude liegen, müssen es aber nicht. Auch in Hinsicht der Nutzer unterscheiden sich verteilte von lokalen Dateisystemen, da die Nutzer sich über eine Schnittstelle von Zuhause in das Netzwerk einloggen und Berechnungen starten können. Im Gegensatz zu parallelen Dateisystemen kann es bei verteilten Dateisystemen zu unterschiedlichen Ergebnissen bei der wiederholten Ausführung von gleichzeitigen Berechnungen kommen, da die Semantik für die gleichzeitige Berechnung teilweise nicht definiert ist.[5]

Welche genauen Zusatzfunktionen ein verteiltes Dateisystem zu einem parallelen Dateisystem machen, ist nicht genau definiert, wodurch es oft zu Unstimmigkeiten in der Literatur kommt. Für diesen Aufsatz werden solche verteilten Dateisysteme, die die

[4] https://en.wikipedia.org/wiki/Parallel_Virtual_File_System [entnommen am 27.02.2019]
[5] R. Sandberg, D. Goldberg et al.: Design and implementation of the Sun network filesystem

Optimierung der parallelen Ein- und Ausgabe sowie der parallelen Berechnung zum Ziel haben, als parallele Dateisysteme bezeichnet.

1.4 Geschichte

Mit dem zwischen 1992 und 1995 entwickelten „Vesta“ oder auch „Tiger Shark“, welches später unter dem Namen „Parallel I/O File System“ veröffentlicht wurde, definierte IBM damit die Geschichte der parallelen Dateisysteme.[6] Ziel war zunächst ein skalierbares und zuverlässiges System für *Video on Demand (VOD)* zu schaffen, das parallel, mit Fehlertoleranz und einem gemeinsamen Namensraum gleichzeitig eine große Anzahl an Videostreams bereitstellen kann. Im Jahr 1998 erschien der Nachfolge „General Parallel File System“ (kurz: *GPFS*), welches POSIX-konform war, aber zunächst nur in Verbindung mit Storage-Hardware vertrieben wurde.[7]

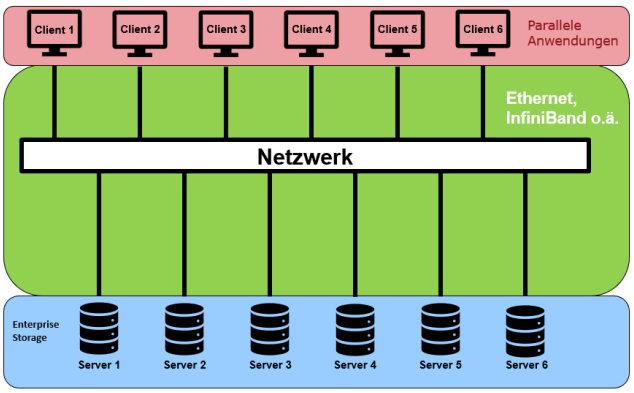
Im Jahr 1999 begann Peter J. Braam die Entwicklung des parallelen Dateisystems *Lustre*, dessen Name sich aus „Linux“ und „Cluster“ zusammensetzt. Sun Microsystems kauften sich 2007 in die von Braam für Lustre gegründete Firma „Cluster File Systems Inc.“ ein. Nach der Übernahme von Oracle Corporation in 2010 folgte am Ende des Jahres die Ankündigung, die Weiterentwicklung von Lustre einzustellen. Daraufhin verließen einige ehemaligen Lustre-Entwickler das Unternehmen und gründeten das Startup Whamcloud, das 2011 die Rechte erhielt, Lustre weiterzuentwickeln und zu warten.[8] Heutzutage ist Lustre der Marktführer unter den parallelen Dateisystemen in der *Top500* des Virtual Institute of IO (Vi4IO).[9]

Neben dem Einsatz im *HPC* haben parallele Dateisysteme in den letzten Jahren immer mehr an Bekanntheit gewonnen als Google sein „Google File System“ für seine Suchmaschine und weitere Anbieter ihre Lösungen für verschiedene Cloud-basierte Dienste vorstellten.[10]

[6] Peter F. Corbett, Dror G. Feitelson et al.: The Vesta parallel file system
[7] https://researcher.watson.ibm.com/researcher/view_group.php?id=4840 [entnommen am 27.02.2019]
[8] https://en.wikipedia.org/wiki/Lustre_(file_system)#History [entnommen am 28.02.2019]
[9] https://www.vi4io.org/io500/start [entnommen am 28.02.2019]
[10] https://ai.google/research/pubs/pub51 [entnommen am 28.02.2019]

2. parallele Ein- und Ausgabe

2.1 Architektur



(Abb. 1: in Anlehnung an Peter F. Corbett, Dror G. Feitelson et al.: The Vesta parallel file system, Figure 1)

Die Architektur der meisten mit einem parallelen Dateisystem arbeitenden Supercomputer ist sehr ähnlich.[11] Die in Abb. 1 im rot unterlegten Bereich befindlichen Rechenknoten sind für das Ausführen von nutzerdefinierten Aufgaben zuständig. Die im blau unterlegten Bereich befindlichen Speicherknoten umfassen ein oder mehrere physische Speichermedien sowie das parallele Dateisystem und stellen für alle Rechenknoten nutzbare Ressourcen bereit.[12] Die Aufteilung auf dedizierte Knoten für Ein- und Ausgabe sowie Berechnungen sorgt dafür, dass bei der auf jede Berechnung folgende Speicherung der Daten der Prozessor nicht untätig auf den Abschluss der Speicherung warten muss. Durch diese architektonische Entscheidung wird sichergestellt, dass die Prozessoren der Rechenknoten möglichst wenig Leerlauf haben und somit die insgesamte Berechnungszeit verringert wird.

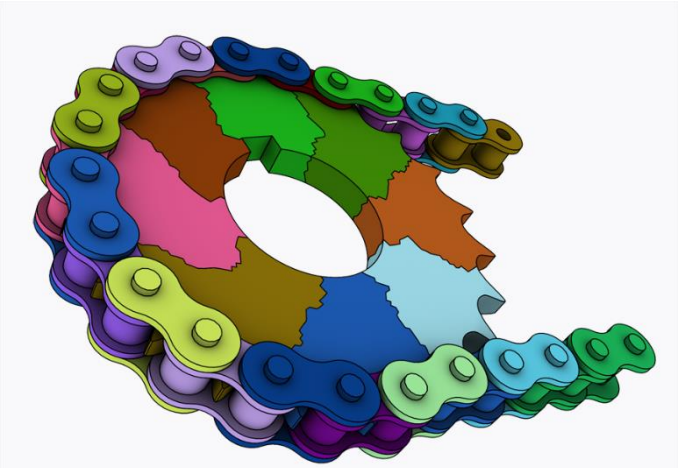
Das Netzwerk (Abb. 1, grün unterlegter Bereich) basiert bei den meisten in der Top500 des Vi4IO auf Ethernet oder InfiniBand[13] und verbindet die Rechen- und Speicherknoten. Die Geschwindigkeit dieser Verbindung ist für die Gesamtgeschwindigkeit des Systems zentral und kann möglicherweise zu einem Flaschenhals werden.

2.2 parallele Anwendungen

Im Gegensatz zu normalen Computern wie sie in jedem Haushalt vorkommen, werden im *HPC*-Bereich nicht sequenziell Berechnungen durchgeführt, sondern parallel.[14] Um eine parallele Berechnung zu ermöglichen wird das Ausgangsproblem zunächst in

[11] Peter F. Corbett, Dror G. Feitelson et al.: Parallel I/O Subsystems in Massively Parallel Supercomputers
[12] Peter F. Corbett, Dror G. Feitelson et al.: The Vesta parallel file system
[13] Erklärung: „Eine Hardwareschnittstelle für die Hochgeschwindigkeitsübertragung auf kurzen Distanzen mit geringer Latenz.“, https://de.wikipedia.org/wiki/InfiniBand [entnommen am 06.03.2019]
[14] Chris Woodford: Supercomputers, 22.12.2018

kleinere Teilprobleme (engl. *sub-domains*) aufgeteilt, die anschließend unabhängig voneinander auf den verschiedenen Rechenknoten berechnet werden können (siehe Abb. 2).[15] Abschließend werden diese Teilprobleme wieder zusammengeführt und die finale Berechnung des Ausgangsproblems kann durchgeführt werden. Diese Vorgehensweisen werden unter dem Begriff *domain decomposition methods* zusammengefasst.



(Abb. 2: http://industry.it4i.cz/en/research/basic-research/feti-based-domain-decomposition-methods/ [entnommen am 6.12.2018])

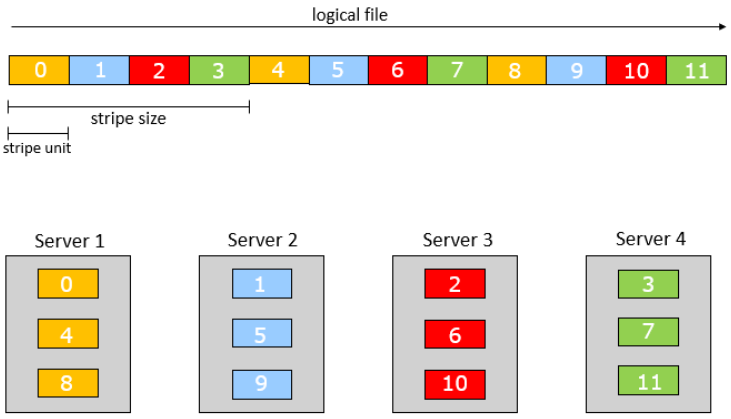
In Verbindung mit der bereits erläuterten Architektur wird somit ein möglichst geringer Leerlauf bei den Rechenknoten erreicht. Es kann jedoch trotzdem dazu kommen, dass ein Rechenknoten keine Berechnungen durchführt. Dies ist der Fall, wenn die Zeitspanne der Berechnung von zwei Teilproblemen kürzer ist, als die Zeitspanne der Speicherung innerhalb des Ein- und Ausgabeknoten. Für die Zeit bis der Ein- und Ausgabeknoten wieder frei ist, muss der Rechenknoten das Ergebnis der zweiten Berechnung im Speicher halten und kann vorerst kein neues Teilproblem lösen.[16] Der zu erwartende Geschwindigkeitsgewinn gegenüber einer seriellen Anwendung lässt sich mit Hilfe vom Amdahlschen Gesetz (engl. *Amdahl's law*) ermitteln.[17]

2.3 Datenspeicherung

Ziel der parallelen Dateisysteme ist im Hinblick auf die Dateispeicherung ist dem Nutzer eine Datei, wie er es von einem lokalen Dateisystem kennt, darzustellen und gleichzeitig eine möglichst effiziente und schnelle parallele Speicherung zu erreichen. Dazu wird eine Datei in „Schnipsel" (engl. *stripes*) zerlegt und auf verschiedene Speicherknoten verteilt – dieser Vorgang nennt sich *striping*. Die Verteilung geschieht dabei meist im *round robin* Verfahren, bei dem jeweils ein stripe auf den Speicherknoten mit dem niedrigsten definierten Index abgelegt wird, der nächste stripe auf den *n+1*-ten

[15] Victorita Dolean et al.: An Introduction to Domain Decomposition Methods: algorithms, theory and parallel implementation, 01.05.2016
[16] Peter F. Corbett, Dror G. Feitelson et al.: Parallel I/O Subsystems in Massively Parallel Supercomputers
[17] https://en.wikipedia.org/wiki/Amdahl%27s_law [entnommen am 01.03.2019]

Speicherknoten abgelegt bis die Datei vollständig abgespeichert wurde. Bei *n* Speicherknoten befindet sich der *i*-te stripe einer Datei auf Speicherknoten *i mod n*. Somit können selbst Dateien gespeichert werden, deren Größe die des größten physischen Speichermediums, das an das System angeschlossen ist, überschreitet. Die Verteilung wird vor den Anwendungen und Nutzern versteckt, sodass diese die Datei lediglich als eine Einheit sehen. Wenn eine Anwendung auf einen Teil einer Datei zugreifen will, ermittelt das parallele Dateisystem den Block in dem sich der, von der Anwendung angefragte, Abschnitt der Datei befindet und stellt ihr diesen zur Verfügung.



(Abbildung 3: Nachzeichnung, Samuel Lang, „Parallel File Systems, IIT Course: Data-Intensive Computing“, September 20, 2010)

Der Vorgang des *striping* kann auch gleichzeitig dazu genutzt werden eine gewisse Fehlertoleranz aufzubauen, indem einzelne *stripes* mehrfach und auf verschiedenen Speicherknoten gespeichert werden. Falls ein Speicherknoten ausfällt, kann auf die Kopie des stripes auf dem anderen Speicherknoten zugegriffen werden und die Datei wiederhergestellt werden.

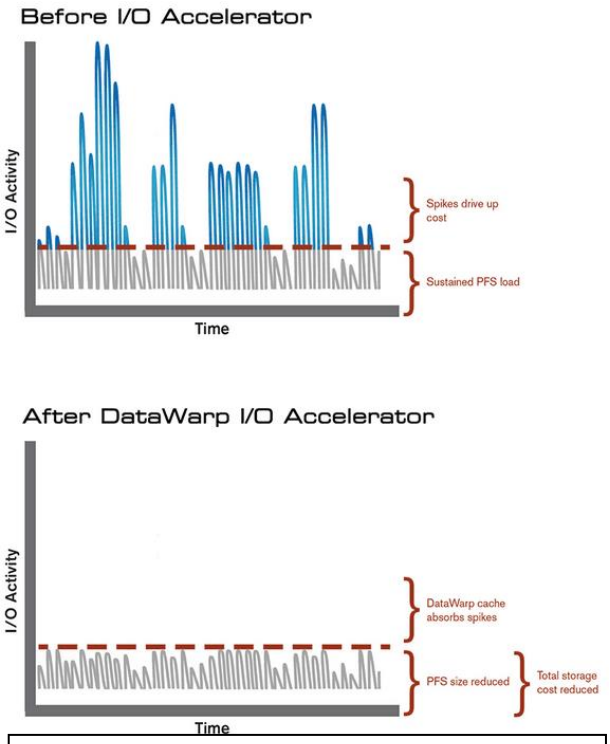
2.4 Metadaten

Metadaten beinhalten wichtige Details zu Dateien und werden bei jedem Lese- oder Schreibbefehl abgerufen.[18] Daher ist ihre Umsetzung kritisch für die Geschwindigkeit eines Dateisystems. Sie werden zunächst in strukturelle Metadaten, die Informationen über den Speicherort (Verzeichnis) beinhalten, und deskriptive Metadaten, die eine Vielzahl an Informationen wie zum Beispiel Zugriffsrechte, den Autor der Datei sowie das Erstellungsdatum beinhalten, unterteilt. Die strukturellen Metadaten in einem parallelen Dateisystem geben an auf welchen Speicherknoten sich die Fragmente bzw. *stripes* der Datei befinden. Ein Fehlen der strukturellen Metadaten zu einer Datei, führt dazu, dass die Datei nicht gelesen werden kann und somit für das System nicht-existent

[18] Vilobh Meshram et al.: Can a Decentralized Metadata Service Layer Benefit Parallel Filesystems?

ist. Falls die deskriptiven Metadaten zu einer Datei fehlen sollten, kann das System zwar nicht feststellen, ob ein Benutzer auch die Rechte besitzt auf die Datei zuzugreifen, aber der eigentliche Inhalt der Datei würde nicht verloren gehen.

Bei der Speicherung der Metadaten gibt es drei zunächst grundsätzlich verschiedene Ansätze. Bei einem lokalen Dateisystem werden Metadaten meist direkt im *header* der Datei und somit auch auf der gleichen Festplatte gespeichert; man spricht von einer direkten Speicherung der Metadaten.[19] Eine direkte Speicherung der Metadaten im *header* der Datei, wie es bei lokalen Dateisystemen üblich ist, ist bei parallelen Dateisystemen nicht möglich, da eine Datei über mehrere Server verteilt gespeichert wird. Die zentrale Speicherung der Metadaten auf einem dedizierten Server wird vor allem in Hochleistungsrechnern eingesetzt, bildet in Hinsicht auf Ausfallsicherheit jedoch ein *Single Point of Failure* und bezüglich der Performance einen Flaschenhals. In der Praxis, wie zum Beispiel beim parallelen Dateisystem Lustre, werden die Metadaten auf mehrere Server aufgeteilt, sodass die Metadaten einer jeden Datei einen dedizierten Server haben und vice versa.[20] Dazu Bedarf es jedoch einer Implementation seitens der Rechenknoten, da diese darüber informiert werden müssen welchen Server sie kontaktieren müssen, um eine bestimmte Datei abzurufen. Alternativ kann dies auch auf eine sogenannter Management Server ausgelagert werden, der dann von allen Rechnerknoten nach dem richtigen Metadatenserver für eine bestimmte Datei gefragt werden kann.



(Abbildung 4: https://www.cray.com/products/storage/datawarp [entnommen am 03.03.2019])

3. Ausblick

Die Entwicklung und Verbesserung leistungsstarker und effizienter Systeme umfasst vor allem die Aufgabe möglichst keine Flaschenhalse im System zu haben, die restliche Komponenten des Systems limitieren. Früher war unteranderem die Prozessorleistung der Flaschenhals großer Systeme. Bei stetig steigender Prozessorleistung wurde jedoch die Ein- und Ausgabe immer mehr zum

[19] https://www.ianus-fdz.de/it-empfehlungen/metadaten-speicherung [entnommen am 20.02.2019]
[20] Andreas Dilger: Lustre Introduction

Flaschenhals der großen Clustersysteme.[21] Parallele Dateisysteme versuchen als Software-Lösung dieser Probleme Herr zu werden, sind jedoch nicht die einzige Möglichkeit große Systeme zu beschleunigen. Eine weitere Lösung dafür sollen *burst buffer* (auch: *I/O Accelerator*) bereitstellen. Wie einst bei den Hauptprozessoren (engl. *CPU*) soll auch bei den Rechen- und Speicherknoten ein schneller *on-board Speicher* (engl. *Cache*) dafür sorgen, dass Spitzen innerhalb der anfallenden Daten abgefangen werden können und die Rechnerknoten möglichst schnell wieder mit der Berechnung anfangen können (Vgl. Abb. 4).
Ein weiterer Gegenstand der Forschung ist die dezentrale Speicherung der Metadaten. Ziel ist die Verteilung der Last bei Dateiaufrufen auf viele Knoten anstelle der herkömmlichen Methode diese Last auf einige wenige Metadaten Server zu leiten. Eine praktikable Umsetzung dieser dezentralen Speicherung der Metadaten gibt es bisher noch nicht.[22]

4. Zusammenfassung

Parallele Dateisysteme bieten für zeitkritische Anwendungen und Hochleistungssysteme eine Lösung zur Optimierung der Ein- und Ausgabe. Vor allem im *HPC*-Bereich werden hohe Datendurchsatzraten benötigt, um eine große Menge an Daten schnell und effizient zu verarbeiten. Mit Hilfe einer gezielten Unterstützung seitens des Dateisystems, wird eine möglichst hohe Parallelisierung der zur Verfügung stehenden Ressourcen ermöglicht. So wird den parallelen Anwendungen ermöglicht eine größtmögliche Auslastung der Rechenknoten zu erreichen bzw. ein Ausbremsen der Rechenknoten durch die Speicherknoten zu verhindern.
Auch die Architektur der Systeme hat einen großen Einfluss auf die Geschwindigkeit der parallelen Dateisysteme. Auf Seiten der Hardware kann mit Hilfe der burst buffer die Software dahingehend unterstützt werden, dass Belastungsspitzen abgefangen werden, indem die burst buffer als Puffer bzw. performante Zwischenablage eingesetzt werden.

[21] https://en.wikipedia.org/wiki/Moore%27s_law [entnommen am 01.03.2019]
[22] Vilobh Meshram et al.: Can a Decentralized Metadata Service Layer benefit Parallel Filesystems?

5. Literaturverzeichnis

1. https://martinvogel.de/lexikon/bottleneck.html [entnommen am 27.02.2019]
2. Markus Ermes: Verteilte, parallele Dateisysteme, 18.12.2018
3. http://www.linux-magazin.de/ausgaben/2007/11/need-for-speed/ [entnommen am 28.02.2019]
4. https://en.wikipedia.org/wiki/Parallel_Virtual_File_System [entnommen am 27.02.2019]
5. R. Sandberg, D. Goldberg et al.: Design and implementation of the Sun network filesystem
6. Peter F. Corbett, Dror G. Feitelson et al.: The Vesta parallel file system
7. https://researcher.watson.ibm.com/researcher/view_group.php?id=4840 [entnommen am 27.02.2019]
8. https://en.wikipedia.org/wiki/Lustre_(file_system)#History [entnommen am 28.02.2019]
9. https://www.vi4io.org/io500/start [entnommen am 28.02.2019]
10. https://ai.google/research/pubs/pub51 [entnommen am 28.02.2019]
11. Peter F. Corbett, Dror G. Feitelson et al.: Parallel I/O Subsystems in Massively Parallel Supercomputers
12. Chris Woodford: Supercomputers, 22.12.2018
13. Victorita Dolean et al.: An Introduction to Domain Decomposition Methods: algorithms, theory and parallel implementation, 01.05.2016
14. https://en.wikipedia.org/wiki/Amdahl%27s_law [entnommen am 01.03.2019]
15. Vilobh Meshram et al.: Can a Decentralized Metadata Service Layer Benefit Parallel Filesystems?
16. https://www.ianus-fdz.de/it-empfehlungen/metadaten-speicherung [entnommen am 20.02.2019]
17. Andreas Dilger: Lustre Introduction
18. https://en.wikipedia.org/wiki/Moore%27s_law [entnommen am 01.03.2019]
19. http://industry.it4i.cz/en/research/basic-research/feti-based-domain-decomposition-methods/ [entnommen am 06.12.2018]
20. Samuel Lang, „Parallel File Systems, IIT Course: Data-Intensive Computing“, September 20, 2010
21. https://www.cray.com/products/storage/datawarp [entnommen am 03.03.2019]