2

Grundlagen

2.1 Klassifikation von Clustersystemen

Bevor auf das komplexe Thema SunCluster 3, hier auf Basis der Version 3.2, eingegangen werden kann, ist zunächst die Begriffswelt Cluster zu definieren und ein Verständnis in das "wofür", "warum" und "warum geht das" zu entwickeln.

Der Begriff *Cluster* beschreibt zunächst einmal eine Menge von Einzelkomponenten, die in einem definierten Bezug stehen. Im Solaris-Umfeld ist beispielsweise der Begriff Cluster auch eine Beschreibung für eine definierte Sammlung von Softwarepaketen, aus denen eine Betriebssysteminstallation erstellt wird.

In der Welt der Multirechnerumgebungen steht der Begriff Cluster für unterschiedliche Strategien und Konfigurationen eines Verbundes von Rechnern. Es wird in der Regel unterschieden in

Performancecluster Dieser Typ Cluster stellt einen Rechnerverbund dar, dessen Aufgabe eine schnelle Abarbeitung von Jobs ist. Dazu werden die Jobs auf einzelne Nodes des Performanceclusters verteilt. Die Verteilung wird in der Regel zentral notiert, und nach Abarbeitung des Jobs wird das Ergebnis bereitgestellt. Typische Vertreter dieser Cluster sind Gridengines und so genannte Compute Farms, welche nicht Gegenstand dieses Buches sind.

Redundanzcluster Dieser Typ Cluster stellt einen Rechnerverbund dar, dessen Aufgabe die hochverfügbare Bereitstellung eines oder mehrerer Dienste bzw. Services oder Dataservices ist. Ein Redundanzcluster wird selten an der Belastbarkeitsgrenze der einzelnen Clusternodes betrieben, da schon eine Überlastung einer Node eine nicht anforderungskonforme Diensterbringung zur Folge haben kann.

Gegenstand dieses Buches sind so genannte Redundanzcluster, im weiteren Cluster genannt. Wie zuvor skizziert, ist die Aufgabe eines Redundanzclusters,

R. Dietze, *SunCluster*, X.systems.press, DOI 10.1007/978-3-540-33806-2_2,

einen Dataservice, etwa eine Datenbank, einen Webserver, oder eine andere beliebige Serverapplikation in jedem Fall für Clientsysteme verfügbar zu halten. In Graphik 2.1 auf Seite 17 sind die an einer Client-Server Applikation beteiligten Komponenten dargestellt. Jede einzelne kann auf unterschiedliche Art und Weise ausfallen. Das Clustersystem als solches ist in dieser Betrachtung nur eine Teilkomponente des Gesamtsystems. Angefangen mit den Clientsystemen, das ist die Dialogschnittstelle des Anwenders eines Dienstes, über die Netzwerkanbindung bis hin zum Dataservice erbringenden Serversystem ist für eine hochverfügbare Anwendung jede Teilkomponente zu betrachten.

Die Verfügbarkeit wird in so genannten Service Level Agreements (SLA) nicht selten an der Dialogstation definiert. Bei der Festlegung der Verfügbarkeit korrelieren die Anschaffungs und Betreiberkosten überproportional zu der Höhe der Verfügbarkeitsanforderung. Es ist zu vermeiden das Clustersystem bei der Planung isoliert zu betrachten, Punkte wie die redundante Auslegung der Netzwerkschicht, der Stromversorgung, der Verfügbarkeit der Dialogsysteme etc. sind ebenfalls in die Planung mit aufzunehmen.

2.2 Techniken und der Begriff der Hochverfügbarkeit

Verfügbarkeiten bei IT-Systemen werden als Anteil Ausfallzeit pro Jahr angegeben. Der Ausfall eines einzelnen Rechners ist bei modernen Rechnersystemen bekannter Serverhersteller wie IBM, Sun Microsystems, Fujitsu und andere mit Verfügbarkeiten von ca. 99.9% recht unwahrscheinlich. Jeder weitere Prozentpunk Verbesserung der Ausfallsicherheit erhöht die Kosten für das Gesamtsystem stark. Hochverfügbarkeit wird vgl. Tabelle 2.1 textuell in sechs AES-Verfügbarkeitsklassen[1], definiert durch die Harvard Research Group (HRG), unterschieden.
Die rein textuelle Beschreibung wie sie die Harvard Research Group vorgenommen hat, trifft zunächst keine Aussage über zulässige Ausfallzeiten. Wie zuvor angemerkt, wird die Verfügbarkeit auch bzgl. der Online- bzw. Ausfallzeiten definiert, wie es in Tabelle 2.2 gezeigt ist. Berechnet wird die Verfügbarkeit entsprechend Gleichung 2.1, wobei die Uptime die Zeit beschreibt, die der Dataservice verfügbar ist und die Downtime die Zeit beschreibt, in der der Dataservice nicht verfügbar ist obwohl er verfügbar sein sollte. Die hier verwendete Downtime ist eine Funktion in Abhängigkeit der Zuverlässigkeit der verwendeten Hardwarekomponenten (Reliability) und der Zuverlässigkeiten des verwendeten Softwarestacks und wird hier nicht näher beschrieben[2].
Die Verfügbarkeit eines Systems oder einer Anlage wird in Klassen unterschieden, wobei die Klasse die Anzahl der Neunen vor und nach dem Komma beschreibt. Hiernach ist beispielsweise die Verfügbarkeitsklasse 3, 99.9% mit 8.76h/a (Stunden Ausfallzeit pro Jahr) angegeben.

[1] Avaliability Environment Classification
[2] Siehe hierzu auch: Sun 806-3345-10

Klasse	Bezeichnung	Beschreibung
AEC-0	Conventional	Funktion unterbrechbar, Datenintegrität nachrangig
AEC-1	Highly Available	Funktion unterbrechbar, Datenintegrität verpflichtend
AEC-2	High Availability	Funktion in festgeleger Zeit unterbrechbar, kurzzeitig
AEC-3	Fault Resilient	Funktion in festgelegten Zeiten ununterbrochen Verfügbar
AEC-4	Fault Tolerant	Funktion muß unterbrechungsfrei verfügbar sein
AEC-5	Disaster Tolerant	Funktion muß unter allen Umständen verfügbar sein

Tabelle 2.1. Verfügbarkeitsklassen nach HRG

$$Availability[\%] = \left(1 - \frac{Downtime}{Uptime + Downtime}\right) * 100 \tag{2.1}$$

Klasse	Verfügbarkeit	Uptime
2	99%	87.66 h/a
3	99.9%	8.76 h/a
4	99.99%	52.6 min/a
5	99.999%	5.26 min/a
6	99.9999%	32 s/a
7	99.99999%	3 s/a

Tabelle 2.2. Verfügbarkeitszeiten im Überblick

Neuerdings werden Virtualisierungstechniken beworben hohe Verfügbarkeiten zu ermöglichen, kann man doch quasi ein Image des virtualisierten Systems schnell neu starten oder gar auch ein Reservesystem migrieren. Die so genannte Hochverfügbarkeit beginnt jedoch erst bei 99.999% Dataservicepräsenz. Für Verfügbarkeitsquoten in dieser Qualität wird jedoch mit Redundanz gearbeitet. Dabei steht das Bestreben eine so genannte Downtime, das ist die Zeit, die eine Applikation nicht verfügbar ist, möglichst kurz zu halten in Opposition zu den dadurch entstehenden Kosten. Es werden dazu folgende Ansätze bzw. Verfahren in Betracht gezogen:

Cold Standby : Die kostengünstigste Methode in Bezug auf Hard- und Softwarekosten. Sie zieht jedoch hohen administrativen Aufwand nach sich. Es wird dabei ein fertig installiertes und konfiguriertes Rechnersystem im abgeschalteten Zustand (Cold) fertig verkabelt und betriebsbereit neben den Produktionsrechner gestellt. Es ist dabei ein Administrator notwendig, der erkennt ob das Produktionssystem ausgefallen ist. Bei Ausfall

muß er sicherstellen, daß das Produktionssystem aus ist und muß das aktuell gehaltene Ersatzsystem mit gleichem Datenstand starten. Fehlerfälle sollten somit nur auftreten, während der Administrator Dienst hat. Anderenfalls ist eine Bereitschaft einzurichten und die längere Ausfallzeit in Kauf zu nehmen.

Hot Standby : Das Ersatzsystem mit aktuellem Daten- und Konfigurationsstand ist bereits in Betrieb, ein Administrator muß die Dataservices lediglich auf dem Produktionssystem stoppen und auf dem Ersatzsystem starten.

Service Monitor : Das Monitoring eines hochverfügbar zu haltenden Dataservices und im Falle eines Fehlers der Neustart des Dataservices durch den Service Monitor ist eine der einfachsten Lösungen, wurde jedoch lange Zeit vernachlässigt. Auf einem einzelnen Server kann durch einen Service Monitor nur der Ausfall des Dataservice behandelt werden. Ein Ausfall von Hardwarekomponenten ist durch den Service Monitor nicht abfangbar.

Diesen Weg geht OpenSolaris. Motiviert aus der langjährigen Clustererfahrung wurde die Utility SMF (Service Management Facility) in das Betriebssystem eingebracht, die clustertypisch auf Basis von Start-/Stop-/Monitorprogrammen einen Service per Kommando startet und stoppt und während der Laufzeit des Programms seine Ausführung überwacht. Bei einer nicht durch die SMF-Utility kommandierten Terminierung des Services geht SMF von einer Fehlersituation aus und startet den Service erneut. Hier wurden Faultmonitormechanismen (Neustart einer Applikation die nicht mehr in der Prozesstabelle ist, Präsenzcheck vgl. Abschnitt 9.2.5.3.3ab Seite 136, PMF), wie sie in der Clusterwelt alltäglich sind, auf ein Einzelsystem gebracht. Hiermit wird die Verfügbarkeit deutlich erhöht. Siehe hierzu auch *OpenSolaris für Anwender, Administratoren und Rechenzentren*, Kapitel 7.1, Service Management Facility.

Loadbalancer : Wenn eine Anfrage von einem Clientsystem kommt, entscheidet ein Loadbalancer welcher Applikationsserver die Anfrage bearbeiten soll. Dabei verteilt ein Loadbalancer die Anfragen entsprechend der Belastung der Serversysteme bzw. anderen vorgegebenen Regeln. Fällt ein Applikationsserver aus, so ist er nicht verfügbar. Für einen Loadbalancer bedeutet dies, daß der nächste Job einem anderen Applikationsserver zuzuteilen ist. Hierbei ist ein zentraler Datastore notwendig, der unter Beachtung von Filelockingmechanismen Schreibrechte an die Applikationsserver vergibt. Typischerweise werden für den Datastore redundant gehaltene Fileserver verwendet, im kostengünstigeren Fall NAS-Devices. Ein typisches Einsatzfeld sind webserverbasierte Dienste wie beispielsweise e-Shop Anwendungen oder Informationsdienste und Web-Fileserver für Softwaredownloads. Die Verwendung von Loadbalancern läßt sich so auch zur Erhöhung der Verfügbarkeit nutzen. Sie stellen eine interessante Alternative zu Clustersystemen dar, wenn die einzelne Transaktion nicht bedeutend im Vergleich zur Gesamtverfügbarkeit

ist. Fällt ein Applikationsserver aus, so wird die nächste Anfrage einem verfügbaren Applikationsserver zugeteilt. Der ausgefallene Applikationsserver schließt die von ihm bearbeitete Transaktion jedoch nicht mehr ab.

Softwarecluster : Softwarecluster führen im Prinzip eine automatische Umschaltung eines Dataservices von dem Produktionssystem auf das Reservesystem durch. Es ist eine Kombination von Hot Standby und Service Monitor, erweitert um Monitoringmechanismen für weitere Komponenten des Servers, wie I/O-Interfaces, Filesysteme, Exportrechte etc. Ein Softwarecluster überwacht den Zustand der Dataservice mit Hilfe eines Service Monitors und wird im Fehlerfall automatisiert, unbetreut und unauffällig einen Neustart oder eine Umschaltung des Dataservices auf die im Hot Standby gehaltene Reservemaschine durchführen. Ein Cluster kann jedoch nur dann einen Dataservice hochverfügbar halten, wenn der Dataservice bei einem Neustart ohne manuelle Eingriffe verfügbar ist. Das applikationseigene Recovery muß in diesem Fall ausreichend sein. Das Thema Softwarecluster wird im folgenden intensiver betrachtet.

Hardwarecluster : Dies sind in der Regel im so genannten Lockstepmodus arbeitende Maschinen. Hierbei führen zwei Rechner exakt die gleichen Programmschritte aus. Nach außen erscheinen sie wie ein System. Softwarefehler können hierdurch jedoch nicht abgefangen werden. Dazu kommt, daß diese Systeme sehr eng beieinander stehen müssen (gleiches Gehäuse, u.U. auf der gleichen Platine) Die Systeme sind erheblich kostenintensiver und bieten ohne einen Dataserverice Monitor keinen Schutz gegen Softwareausfälle.

Die Verfügbarkeit wird im allgemeinen an der Dienstschnittstelle betrachtet. Die Dienstschnittstelle ist das Dialogsystem. Im folgenden werden daher Abhängigkeiten und Redundanzkonzepte bei Dialogsystemen und anschließend die Möglichkeiten zur Erhöhung der Verfügbarkeiten der Backendserver auf Hardware- und Softwareebene skizziert.

2.3 Einsatzgebiete von Clustersystemen

Selten ist dem Benutzer an einem Frontendsystem bekannt ob die angeforderten Dienste hochverfügbar bereitgestellt werden oder nicht, sie werden genutzt. In modernen EDV-gestützten Umgebungen können Rechnerausfälle jedoch weitreichende Folgen haben. Man stelle sich den Ausfall eines Buchungs- und Handlungsrechners einer Bank während der aktiven Handelszeit an der Börse, oder den Ausfall des zentralen Leitrechners zur Steuerung einer lebenswichtigen betriebstechnischen Anlage vor. So unterschiedlich die Einsatzgebiete sind, so auffällig ist, daß ein Rechnerausfall hohe finanzielle Verluste oder

im Extremfall Personenschaden zur Folge haben kann. In solchen Einsatzbereichen gelten Kostenargumente selten, der ausfallsbedingte Schaden ist zu hoch.

Die Forderung nach hochverfügbaren Diensten kommt vermehrt auch in weniger kostspieligen Bereichen auf, denn der Ausfall eines zentralen File-, Datenbank- oder Applikationsservers hat in der Regel zur Folge, daß Mitarbeiter eines Unternehmens bei Ausfall von Rechnersystemen nicht produktiv arbeiten können. Diese Betrachtung trifft für Unternehmensapplikationen ebenso zu wie für Lagerverwaltungssysteme, Telefon- und Serviceprovider etc., für die ein Ausfall eine Einbuße an verkaufter Ware, entgangenem Traffic etc. bedeutet.
Die Notwendigkeit für den Einsatz eines Clustersystems ergibt sich aus:

- der Bilanz der Kosten für ein Clustersystem in Relation zu den Kosten die bei Ausfall eines Rechners entstehen würden,
- den Verfügbarkeitsanforderungen der Nutzer des angebotenen Dienstes, definiert in so genanten Servicelevelagreements,
- Verfahrensbeschreibungen, unternehmesbedingen Eigenauflagen,
- behördlichen Vorgaben oder gesetzlichen Bestimmungen.

2.4 Hochverfügbarkeit im 2-Tier Businessmodell

Ein Anwender, der an einer Dialogstation sitzend auf einen Dataservice zugreift, macht sich in der Regel keine Gedanken über sein Tun. Er ruft eine Applikation auf, die ihrerseits eine Verbindung zu einem Dataservice auf einem Backendserver aufbaut. Damit ist eine Kette von Komponenten entstanden, die bei der Definition der Verügbarkeit einer Anwendung im einzelnen betrachtet werden müssen. Eine Client-Server-basierte Anwenderapplikation kann genau dann nicht mehr arbeiten, wenn auch nur eine Teilkomponente, Dialogstation, Netzwerk, Server, Plattensysteme, Dataservice, ersatzlos ausfällt.
Zur Erreichung eines hochverfügbaren Gesamtsystems sind alle Ebenen einzeln zu betrachten und auf ihre Redundanz, Ausfallwahrscheinlichkeit und ihre Recoveryzeiten zu betrachten. Eine isolierte Betrachtung des Backendsystems ist unter Umständen nicht ausreichend. In die Betrachtungen gehören die Frontendsysteme bzw. Dialogstationen oder Endgeräte, das Netzwerk zwischen Frontend und Backendsystemen, das Netzwerk zwischen den Backendsystemen und das Storagesystem der Backendsysteme. Zwischen Frontend und Backend können Komponenten wie Firewall, Loadbalancer etc. verborgen sein, die ggf. ebenfalls auf ihre Verfügbarkeit hin zu evaluiern sind.
Gewissermaßen von vorne nach hinten betrachtet, ist das erste Glied in der Kette das Dialogsystem, über das auf den hochverfügbaren Dataservice zugegriffen wird. Fällt die Dialogstation aus, ist letztendlich der Dataservice nicht verfügbar. Eine zusätzliche Gefahr für den Dataservice ergibt sich unter Umständen aus der Art des Frontendsystems selbst.

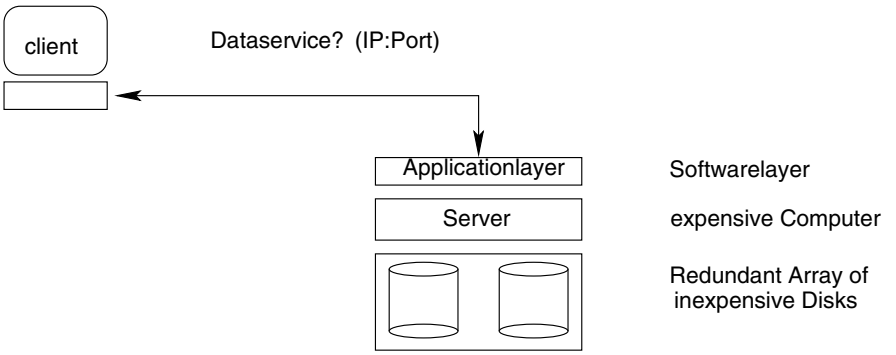

Abb. 2.1. Client-Server Kommunikation

Abb. 2.2. Client-Server Datastore

Wird als Frontendsystem ein so genannter Fatclient, ein voll- und eigenständiger Desktoprechner mit eigenständiger Betriebssysteminstanz, möglicherweise auch noch durch den Benutzer administrativ konfigurierbar, verwendet, so birgt dieser Fatclient nicht nur die Gefahr einer administrativen Monsteraufgabe bei entsprechend vielen Fatclients. Es ist vgl. Graphik 2.2 zu evaluieren wo die Daten der Applikation bei der aktiven Nutzung bzw. Bearbeitung liegen. Oftmals hält die Frontendapplikation, die auf dem Fatclient laufend die Nutzung des Dataservice bietet, Teile der aktiven Daten lokal vor. Der Datastore liegt in diesem Fall nicht nur auf dem hochverfügbaren Backendsystem, sondern auch auf dem Frontendsystem. Ein Ausfall des Frontendsystems würde in diesem Fall zwangsläufig zu inkonsistenen Daten führen obwohl das Backendsystem hohen Verfügbarkeitsansprüchen genügt, denn der Teil der Daten, der noch auf dem Frontendsystem gehalten wurde und vor dessen Ausfall nicht auf das Backendsystem zurückgeschrieben wurde, ist in der Regel unwiederbringbar verloren. Die Daten haben den Backendserver unter Umständen nie erreicht um zur Verarbeitung zu kommen. Dazu kommt, daß das fatclientbasierte Frontendsystem u.U. nicht in der administrativen Hoheit des Betreibers des Backendsystems liegt und daher seitens des Backendsystems keine Datenkonsistenz garantierbar ist.
Ist der Einsatz von Fatclients an dieser Stelle unabdingbar, sollte auf ein durchdachtes Software- und Konfigurationsdeployment zurückgegriffen werden. Dennoch besteht eine große Abhängigkeit sowohl der Gesamtverfügbarkeit als auch der Datenintegrität von der fehlerfreien Funktion der Fatclient Dialogstation.

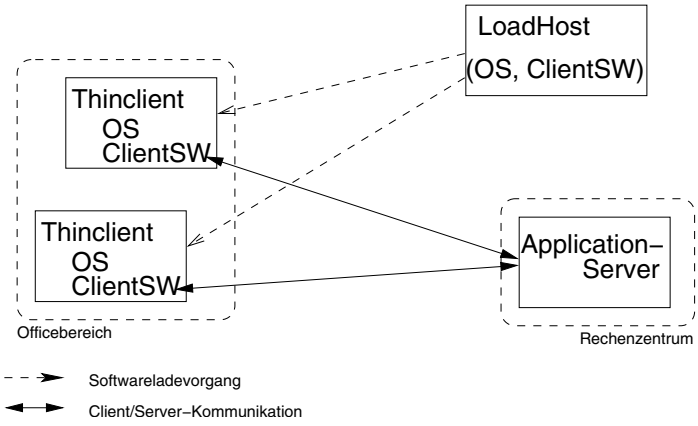

Abb. 2.3. Linux-/Windows-basierter Thinclient

Alternative Ansätze ergeben sich aus der Verwendung so genannter Thinclients als Desktopdialogsystem. Thinclients bieten zwei Vorteile, minimale und serverzentrisch kontrollierbare Systemkonfigurationen der Frontendsysteme und sie halten nur einen minimalen Teil an Funktionalität lokal vor. Der Datastore liegt damit im Betrieb nicht oder nur Teilweise auf Frontend- und Backendsystem vor. Bei Thinclients ist zu unterscheiden in Windows-basierte, Linux-basierte und JVM (Java Virutal Machine) basierte Thinclients. Zu Thinclients werden auch die klassischen X-Terminals zugeordnet, auf deren Basis die JVM-Thinclients von Sun entwickelt wurden.

Thinclient Systeme bieten deutlich mehr Schutz vor Eingriffen. Jedoch ist bei den handelsüblichen Windows und Linux basierten Systemen immer noch eine Bindung des Anwenders auf eine bestimmte Arbeitsstation während einer Sitzung gegeben. Fällt die Dialogstation aus, so fällt damit auch die Clientsoftware in einem undefinierten Zustand aus. Die Datenkonsistenzprobleme der Applikationsdaten, die durch im laufenden Betrieb ausgefallener intelligenter Dialogsysteme entstehen, bleiben bei auf Windows oder Linux basierten Thinclients bestehen. Der verbleibende Vorteil ist die einfache und leicht kontrollierbare Administration der Frontendsysteme und deren leichte Ersetzbarkeit bei Ausfall und damit die Verfügbarkeit eines Dialogsystems als solches. SunRay Thinclients bieten hier einen entscheidenden Vorteil, denn die Abhängigkeit auf eine bestimmte Dialogstation entfällt bzw. wird auf den SunRay Server verlegt. Die SunRay Architektur beschreibt ein serverbasiertes Desktopcomputing. Um hier eine klarere Übersicht zu bieten sind Thinclientsysteme weiter zu unterscheiden in:

Zustandsabhängige Thinclients: Anwendung und Betriebssoftware werden von einem Loadhost geladen und lokal ausgeführt. Die Clientsoftware läuft

lokal auf dem Thinclient. Verbindungen zum Backendserver verlaufen vom Thinclient zum Backendserver. vgl. hierzu Abbildung 2.3

Zustandsfreie Thinclients: Der Thinclient stellt nur das Anzeigesystem (Fensteroberfläche, Tastatur-/Mauseingaben). Die Clientsoftware läuft nicht auf dem Client sondern auf dem Thinclient-Loadhostserver. vgl. hierzu Abbildung 2.4

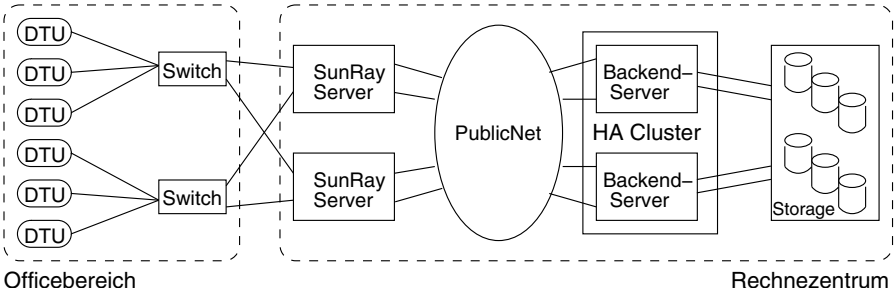

Abb. 2.4. SunRay-basierter Thinclient

Thinclients, egal welcher Klasse, setzen immer einen lauffähigen Server, nach Möglichkeit den Anforderungen des Gesamtsystems entsprechend redundant ausgelegt, voraus. Zur weiteren Beschreibung und Klassifizierung der Typen der Thinclients und deren systemische Einbettung sei hier auf *OpenSolaris für Anwender, Administratoren und Rechenzentren*, Kapitel 8.1, Topologie der Arbeitsumgebung hingewiesen.

Die Verwendung von zustandsfreien Thin Clients, wie es das SunRay System darstellt, bietet die Vorteile einfachster Administration, Zugriffsschutz und Sicherheit, gepaart mit Frontendredundanz auf der Ebene der SunRay Server selbst. Sollte eine so genannte SunRay-Appliance, eine Thinclient Dialogstation, ausfallen, so kann die laufende Arbeitssitzung auf eine andere SunRay Appliance verlagert werden. Alle noch nicht serverseitig verarbeiteten Daten bleiben auf dem aktuellen SunRay Server erhalten, womit die Datenintegrität bei Ausfall einer Dialogstation (DTU) auf der Client- bzw. Frontendseite gewährleistet ist. In Abbldung 2.4 sind zwei Gruppen von Dialogstationen skizziert. Sollte ein Switch im Frontendbereich ausfallen, so fallen damit nicht automatisch alle Arbeitsplaetze aus.

Das SunRay Konzept bietet im weiteren die Option einer redundanten Auslegung der SunRay-Server mit dem zusätzlichen Vorteil einer Lastverteilung. Die redundante Auslegung der SunRay-Server bietet bei Ausfall eines SunRay-Servers in der Redundanzgruppe den Vorteil, daß die verbleibenen redundanten SunRay-Server den Desktopservice übernehmen. Der Anwender hat so in jedem Fall einen funktionierenden und somit verfügbaren Desktop, die

auf dem ausgefallenen SunRay-Server laufende Clientsession ist jedoch verloren.
Zusammenfassend läßt sich sagen:

Fatclient Die Verwendung intelligenter eigenständiger Dialogstationen bieten den Vorteil hoher Versatilität und Rechenleistung auf dem Desktop, verbunden mit den Nachteilen eines unsicheren nicht USV-gestützten Betriebes des Dialogsystems im Officeumfeld, bei dem ein Desktop u.U. benutzerseitig administriert wird, (versehentlich) abgeschaltet oder unmotiviert rebootet wird. Im Officeumfeld ist die Netzwerkverkabelung eher unsicher weil offen, dazu kommt, das Desktop-/Deskside Rechner einer geringeren Zuverlässigkeitsklasse angehören, meist nur ein Netzteil, Hardwarekomponenten können oftmals nur bei abgeschaltetem Fatclient getauscht werden etc. Die Kommunikation verläuft vom ungeschützten Officeumfeld direkt zum redundanten Backendsystem, was unter anderem auch Sicherheitsbedenken entstehen läßt. Fatclients sind Administrationsintensiv und damit in der Regel fehlerbehafteter.

Zustandsabhängiger Thinclient Thinclients, basiert auf einem linux- oder Windows-basiertem Minimalsystem, bieten den Vorteil eines Standardfrontends und vereinfachter und zentral kontrollierbarer Administration. Die Kommunikation verläuft auch hier zwischen dem Desktopsystem und dem redundanten Backendsystem, die hohen Risiken der Nutzung eines Fatclients greifen somit auch hier.

Zustandsfreier Thinclient Hier am Beispiel SunRay skizziert und in der Kürze unvollständig, bietet das Konzept eine Reihe von Vorteilen. Zunächst, sollte eine Desktopeinheit ausgeschaltet werden, so schaltet man sie wieder ein, die DTU startet und anschließend arbeitet man mit allen laufenden Applikationen und Netzverbindungen weiter, der Reboot der DTU hat keinen Einfluss auf die auf dem Desktop laufenden Prozesse. Dazu kommt, daß die Client-Backendkommunikation in den Backendbereich, serverbasiert, verlegt ist. Der SunRay-Server, im geschützten RZ-Bereich arbeitend, kommuniziert direkt mit dem hochverfügbaren Backendsystem. Der Einsatz eines SunRay-Servers, abgesetzt von den Desktopsystemen, erlaubt die Verwendung von Hardware einer höheren Zuverlässigkeitsklasse, was in der Regel bei redundanten Netzteilen anfängt und je nach Auslegung des Serversystems bis hin zu der Möglichkeit der technischen Wartung im laufenden Betrieb gehen kann. Die Administration vereinfacht sich auf die Administration einer einzelnen Maschine, dem SunRay Server. Die einzelnen DTUs erzeugen in der Regel keinen Mehraufwand. Der Ausfall einer DTU läßt sich ggf. ablauftechnisch regeln, indem beispielsweise ein Benutzer seine ausgefallene DTU gegen eine funktionierende DTU an einer Materialausgabe tauschen kann und diese selbst Aufstellen und in Betrieb setzen kann.

In der Praxis ist zu sehen, daß auch bei mehrfachem hardwareseitigen Upgrade der SunRay Server auf aktuellen Leistungsstand, die Unternehmen vielfach

noch mit SunRay Appliances aus der ersten Generation arbeiten, inzwischen wurden allenfalls die Monitore durch TFT-Displays getauscht, ggf. wurden die alten USB-Mäuse gegen modernere Mäuse mit Scrollrad getauscht. Dies führt zu einem in der IT-Welt bemerkenswerten Lebenszyklus für Desktopsysteme von ca. 10 Jahren.

Das nächste Glied in der Kette ist die Netzwerkverbindung zwischen Clientsystem und Backendserver. Hier sollte entsprechend der Anforderungen auf eine ausreichend redundante Verbindungsauslegung im lokalen oder WAN-Bereich geachtet werden. Das vorletzte Glied in der Kette ist ein ausfallsicherer Backendserver mit ausreichend ausfallsicheren Plattensystemen. Zum Schluß ist auf ausreichend ausfallsichere bzw. redundante Stromversorgung, Klima etc. zu achten, was hier nicht Gegenstand der Diskussion sein soll.
Was kann auf dem Backendsystem die Bereitstellung eines Dataservices beeinträchtigen?

Hardwareausfall Ein den Ausfall von Hardwarekomponenten, die
- eine Einschränkung oder einen Ausfall der Kommunikation zwischen Client und Server darstellen,
- eine Einschränkung oder Ausfall des Datastores für die Serverapplikation darstellen,
- ein Ausfall oder Defekt von Teilen des Serversystems oder des Serversystems im Ganzen darstellen.

Softwareausfall Ein Ausfall der serverseitigen Applikation, des Dataservice selbst, bedingt durch Absturz, Verklemmung oder logischer Fehler.

Hier erfolgt die erste Einschränkung: Ein Clustersystem kann keine logischen Softwarefehler auflösen. Es kann eine Serversoftware terminieren und neu starten. Alle softwareseitigen Fehler die durch einen Neustart des Dataservice oder Teilen des Dataservice nicht auflösbar sind, können durch ein Clustersystem nicht aufgelöst werden.

Es ist Aufgabe eines Clustersystems einen Dataservice hochverfügbar zur Verfügung zu stellen. Für ein Clientsystem sollte ein Ausfall bestenfalls nicht bemerkbar sein. Als akzeptabel wird ein Reconnect und Recover, ein Neuverbinden des Clientsystems mit dem hochverfügbar gehaltenen Dataservice der Server im Clusterverbund angesehen. Die hohe Verfügbarkeit wird erreicht, indem eine Monitorsoftware auf Rechnern, die in einem Clusterverbund stehen, im Falle eines Ausfalls von Teilen des Clustersystems Maßnahmen ergreift, um den Dataservice weiterhin für das Clientsystem bereitzustellen. Im Rahmen der Betrachtung von Clustersystemen wird eine mit einem Betriebssystem ausgestattete Clusternode als eine beliebig ersetzbare Teilkomponente angesehen. Ein gezieltes Rebooten einer Clusternode durch den Clusterframemanager ist ein normaler Betriebszustand eines Clustersystems bei Unklarheiten in der Diensterbringung.

Um eine Erklärung zu bieten, warum ein Clustersystem hohe Verfügbarkeiten liefern kann, ist die Art der Kommunikation zwischen Clientsystem und

Backendsystem zu betrachten. Ein Clientsystem auf dem ein beliebiges Betriebssystem läuft, verbindet sich unter Zuhilfenahme eines standardisierten Applikationsprotokolls mit einem Dataservice unter Benutzung der IP-Adresse des Serversystems und mit einer Portnummer, unter der der angeforderte Dataservice auf dem kontaktierten Backendserver zur Verfügung steht. Dem Clientrechner ist dabei nicht bekannt, daß es mit einem Backendserver in einem Clusterverbund verbunden ist, noch ist dem Clientsystem bekannt, welches Betriebssystem auf den Backendsystemen läuft. Das Clientsystem hat keine Kenntnis über Struktur, Aufbau, und Funktion des Backendsystems, es kennt lediglich die IP-Adresse und die Portnummer des angeforderten Dienstes und das Applikationsprotokoll, beispielsweise NFS HTTP oder Datenbankverbindungen. Dabei wird unterschieden in

Stateless Connection Zustandslose Verbindung.
: Hier wird für jede Anfrage und jeden Schreibzugriff die Verbindung erneut aufgebaut. Ein typischer Vertreter ist der HTTP Dienst.

Stateful Connection Verbindungsorientierte Anfrage.
: Hier wird die Verbindung vom Client zum Server aufgebaut und über die gesamte Zeit der Nutzung des Dataservice aufrecht erhalten. Nach Beendigung der clientseitigen Applikation wird die Verbindung in der Regel wieder abgebaut. Typische Vertreter sind Datenbankverbindungen.

Wenn nun im laufenden Betrieb ein Dataservice ausfällt, so sollte es der Client nicht merken oder in der Lage sein, neu zu verbinden. Es ist beispielsweise für einen NFS-Client nicht unterscheidbar, ob ein NFS-Server extrem langsam ist oder ob er gerade heruntergefahren und neu gestartet wurde. Bei Statefull Connections sieht das nicht ganz so einfach aus.
Da ein Clientsystem keinen Einblick in Aufbau, Struktur und Funktion des Backendservers hat und zur Anfrage lediglich IP-Adresse und Port des Dataservice kennt, kann beispielsweise ein Server heruntergefahren werden, die gesamte für den betreffenden Dataservice notwendige Konfiguration auf einen anderen Server verbracht werden und dieser zweite Server gestartet werden. Nach fehlerfreiem Start sollte dieser Server den betreffenden Dataservice unter der gleichen IP-Adresse wieder so anbieten, wie auf dem Ursprungssystem. Der Client bekommt diesen "Serverumzug" nicht mit.

Wenn nun unter einer dem Dataservice zugeordneten IP-Adresse und der angefragten Portnummer ein Dataservice antwortet, der durch mehrere redundante Server die in einem gemonitorten Verbund stehen, zur Verfügung gestellt wird, so ist dieser Dataservice höher verfügbar, als wenn er durch ein Einzelsystem angeboten wird. Aufgabe des Verbundes aus mehreren Einzelnodes ist es dafür Sorge zu tragen, daß die Dataserviceschnittstelle mit dem darunterliegenden Dataservice eindeutig bereitgehalten wird, und im Falle eines Ausfalls für den Client verborgen, der Dataservice automatisiert auf eine andere Node im Clusterverbund, auf eine Redundanznode übertragen wird.

Das dataserviceanfragende Clientsystem greift damit nicht mehr explizit auf ein Backendserver bzw. den von ihm angebotenen Dataservice zu, son-

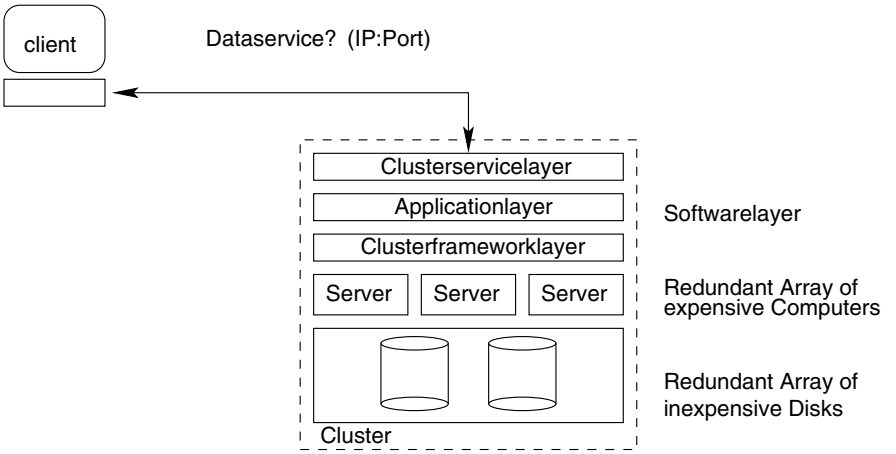

Abb. 2.5. client-cluster

dern auf einen generischen Clusterservicelayer, der als Mittel der Redirektion der Clientanfrage ein Umhängen einer logischen IP-Adresse die dem Dataservice zugeordnet ist, auf eine Redundanznode im Clusterverbund. Bei Stateless Connections ist das Umsetzen der IP-Adresse des Dataservice auf eine Redundanznode vollkommen unkritisch, bei Statefull Connections muß das Clientsystem einen Reconnect ausführen. Das Clientsystem greift somit auf einen Servicelayer zu, der vor ihm verbirgt, daß ein Dataservice auf einer Node gestoppt und auf einer Redundanznode wieder gestartet wird. Der wesentliche Trick hierbei ist, daß als erstes die dem Dataservice zugeordnete IP-Adresse auf der Servernode dekonfiguriert wird. Damit antwortet auf die Clientanfragen keine Node mehr. Ab diesem Zeitpunkt kann das Clusterframework alles Notwendige veranlassen um den betreffenden Dataservice auf einer Redundanznode zu starten und nach erfolgreichem Dataservicestart die Dataservice-IP-Adresse auf der Redundanznode zu aktivieren. Es läßt sich ein durch das Clusterframework gesteuertes Umschalten eines Dataservice von einer Node auf eine Redundanznode skizzieren wie folgt:

1.	Servicenode:	Dekonfiguration der Dataservice-IP-Adresse
2.	Servicenode:	Stopp des Dataservice
3.	Servicenode:	Stopp aller Supportservices des Dataservice
4.	Transition:	Übertragung aller notwendigen Resourcen
5.	Redundanznode:	Start aller Supportservices des Dataservice
6.	Redundanznode:	Start des Dataservice
7.	Redundanznode:	Konfiguration der Dataservice-IP-Adresse

Tabelle 2.3. HA Switch

Hierbei ist zu beachten, daß die Dataservices tatsächlich gestoppt werden und nicht suspended werden. Sie müssen folglich auf der Redundanznode neu gestartet werden. Sollte eine solche HA-Übernahme aus einer Fehlersituation heraus erfolgen, ist nicht immer sichergestellt, daß der Dataservice ordnungsgemäß terminiert wurde. Dies stellt gleichzeitig eine harte Bedingung an clusterfähige Software: Eine in einer Clusterumgebung eingesetzte Serversoftware muß in der Lage sein sich aus einer harten Terminierung (kill-Signal) selbstständig in kurzer Zeit zu recovern, derart, daß keine Dateninkonsistenz vorliegt und der Dataservice seinen Dienst in relativ kurzer Zeit, inklusive aller Recoveryoperationen, wieder aufnehmen kann. Die Übernahmezeiten liegen in Bereichen bis zu ein bis zwei Minuten. Es ist auch zu beachten, daß das Umschalten des Dataservice von Servicenode auf Redundanznode in der Regel zeitlich durch das Clusterframework überwacht wird. Dauert die Umschaltung zu lange, reagiert das Clusterframework auf die Zeitschrankenüberschreitung durch einen erneuten Dataservicerestart, Failover oder Failback. In welcher Art und Weise und wann ein Cluster auf die Zeitüberschreitung reagiert, ist von der Implementation des Clusters, also herstellerabhängig, oder in einigen Fällen sogar benutzerseitig konfigurierbar.

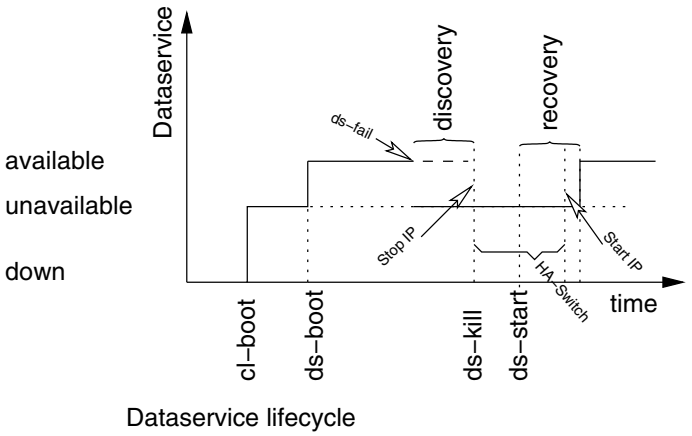

Abb. 2.6. Dataservice Lifecycle

Der in Tabelle 2.3 skizzierte Ablauf einer Migration eines Dataservice von der primary Node auf die secondary Node bedingt auch den Lifecycle eines Dataservices in einem Clusterverbund, detaillierter veranschaulicht in Abbildung 2.6. Mit Start der Clustersoftware werden zunaechst alle konfigurierten Dataservices gestartet. Während des laufenden Betriebes überprüft die Clustersoftware ob der gestartete Dataservice läuft, auf Anfragen antwortet bzw. verfügbar ist. Fällt nun der Dataservice aus, so kann das Cluster erst reagieren wenn der Fehlerzustand erkannt wurde. Folglich wird der Dataservice

"angepingt" bzw. getestet, es wird ein kurzer Moment gewartet. Reagiert der Dataservice während der Wartezeit nicht, so wird er herruntergefahren, klappt dies nicht wird er terminiert. In diesem Moment des fehlerfreien Erkennens des Ausfalls des Dataservice wird die Netzwerkkommunikation zu diesem Dataservice unterbrochen, am einfachsten durch ein Dekonfigurieren der IP-Adresse über die der Dataservice erreicht werden kann. Diese Zeitspanne ist die Discoveryphase, in der der eingetretene Fehlerfall noch nicht aufgelöst wird sondern erst erkannt werden muß. Nach erfolgter Dekonfiguration der IP-Adresse werden die notwendigen Resourcen auf die Redundanznode transferiert, der Dataservice gestartet und anschließend die IP-Adresse über die der Dataservice erreicht werden kann, auf der Redundanznode konfiguriert. Mit dem Start des Dataservice wird der Dataservice u.U. ein Recovery des eigenen Datastores durchführen müssen, da der Dataservice bei der letzten Umschaltung u.U. nicht ordnungsgemäß gestoppt wurde. Wann immer die Startmethode des Dataservice den Service frei gibt, wird die Service-IP-Adresse konfiguriert. Die Bereitstellung der Netzwerkkommunikation kann zeitlich vor Ende der Abarbeitung der Recoverymethoden des Dataservice liegen, wenn der Dataservice dies so zulaesst. In der Zeit, in der die IP-Adressen von Dataservices dekonfiguriert werden bis zu dem Zeitpunkt zu dem die IP-Adressen wieder konfiguriert sind, ist der Dataservice nicht verfügbar. Dieser Zeitraum ist der, in dem der eigentliche HA-Switch, die Übertragung des Dataservice durch die Clustersoftware, durchgeführt wird.

Damit nochmals in Zusammenfassung die Anforderungen an einen clusterfähigen Dataservice:

- Die Datenkonsistenz ist vollständig durch den Dataservice zu gewährleisten, insbesondere bei nicht ordnungsgemäßer Beendigung und anschließendem Neustart des Dataservice.
- Der Dataservice kann ausschließlich nur über eine ihm zugewiesene Service-IP-Adresse genutzt werden (reiner Client-Server Betrieb)
- Der Dataservice darf nicht interaktiv auf dem Server genutzt werden (können)
- Der Dataservice muß sich terminieren lassen, der Mountpunkt des Datastorefilesystems im Shared Storage muß freigegeben sein.
- Ein verklemmter Dataservice muß durch einen Neustart in eine definierte Situation gelangen, aus der heraus der Dienst verfügbar ist. (der Dataservice muß durch einen Neustart "repariert" sein)
- Misslingt ein Stopp oder gar die Terminierung des Dataservice wird u.U. die Node auf der der Dataservice nicht terminierbar ist per Reset aus dem Clusterverbund geworfen. Auch an den Dataservice ist die Anforderung gestellt mit dieser Situation umgehen zu können und den Datenstand konsistent und integer zu halten.

Eine weitere wesentliche Komponente eines Clustersystems ist der im Shared Storage des Clusterverbundes liegende Datastore. Der Datastore ist der für alle Nodes eines Clusters sichtbare Festplattenbereich in dem die Appli-

kationsdaten und ggf. die Applikationen selbst liegen. Es ist für ausreichende Redundanz zu sorgen und für Kommunikationspfade zu den Nodes, die am Clusterverbund teilnehmen.

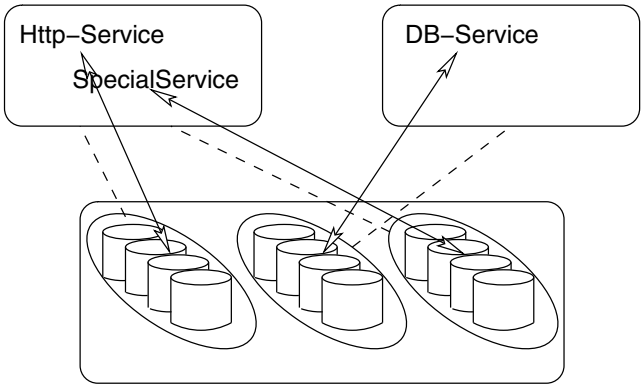

Abb. 2.7. Datastor per Service

Typischerweise geht die jeweilige Clustersoftware davon aus, daß die Redundanz und Verfügbarkeit des Shared Storage durch einen gesonderten Softwarelayer geregelt ist, was gleichzeitig bedeutet, daß die Clustersoftware keine Konsistenz- oder Integritätsprüfungen im Shared Storage durchführt die über einen Filessystemcheck bei Import und Mount des jeweiligen Filesystemes hinausgehen. In der Regel werden bei Clustersystemen SoftRAIDsysteme eingesetzt, da sich das Clusterframework der Import- und Exportfunktionen des jeweiligen SoftRAIDs bedient um die RAIDVolumes von Clusternode zu Clusternode zu transferieren. Bei einem HA-Switch wie zuvor dargestellt, wird dann lediglich der Zugriff auf die Festplatten des Datastores geregelt, der für die Serverapplikation von Belang ist, was die Definition eigener SoftRAID-Volumes für jede einzelne hochverfügbar zu haltende Applikation, insbesondere bei Nichtnutzung des PxFS, mit sich zieht. Je nach Auslegung ist auch eine feste Zuordnung des Datastores eines Dataservice auf jeweils eigene Plattensysteme sinnvoll. Eine solche Umgebung wäre genauer zu planen.

Wird hingegen der gesamte Datastore aller clusterweit konfigurierten Dataservices in die gleiche Volumegruppe gelegt, so können sowohl die Dataservices als auch die Devicegruppen nicht mehr lastverteilend über die am Clusterverbund teilnehmenden Nodes verteilt werden. Daraus ergibt sich u.U. ein wenig flexibles Verhalten des Clusters und möglicherweise Deadlocksituationen bei Teilausfällen bzw. Ausfällen von Subkomponenten.

Nach dieser kleinen Schnelleinführung in die besonderen Umgebungsbedingungen von Clustersystemen, einen Einblick in die Arbeitsweise eines Redundanzclusters, als auch in die Anforderungen an die hochverfügbar zu hal-

tenden Dataservices ist zunächst genauer zu analysieren wie zu erkennen ist, ob ein Dataservice ordnungsgemäß arbeitet und welche Komponenten eines Clusters bei einer Migration eines ausgefallenen Dataservices auf eine Redundanznode bzw. bei der Überwachung des ordnungsgemäßen Betriebes des Gesamtsystems welche Aufgaben hat. Zunächst ein paar Überlegungen zur Überwachung des Systems, denn Fragen wie wie kann festgestellt werden ob ein Programm läuft und funktioniert, wie kann eine sichere Aussage getroffen werden auf dessen Basis ein gewaltsames Terminieren eines Programms durchgeführt wird, nur um es anschließend neu zu starten, sind schwer zu beantworten und doch von existentieller Relevanz in Hochverfügbarkeitsclusterumgebungen.

http://www.springer.com/978-3-540-33805-5

SunCluster
Serververfügbarkeit unter Solaris
Dietze, R.
2010, XXIV, 593 S. 188 Abb., Hardcover
ISBN: 978-3-540-33805-5