



PREDICTABLE PERFORMANCE

Een 'cookbook' van de NAF werkgroep performance prediction

Auteurs: NAF werkgroep performance prediction
Filenaam: NAFkookboekNed.doc
Versie: 1.0
Datum verslag: 13-12-2004 4:03

Copyright (C) 2004 NAF Werkgroep Performance Prediction
All Rights Reserved

Distributielijst:

Naam	Organisatie

Inhoudsopgave

Inleiding	3
Leeswijzer, de structuur van dit document	3
Document historie	3
De werkgroep.....	3
Samenvatting.....	4
Over de noodzaak van performance.....	6
Typering van applicaties	7
Kwantificering van workload	9
Kwantiteiten	9
Soorten gebruikers.....	10
Soorten transacties.....	11
Systeem belasting profiel.....	12
Performance Engineering (PE).....	13
Kwantitatief Analysemodel (ArchiMate)	14
Schattingen CPU Verbruik & Doorlooptijd.....	16
Referentieschattingen.....	16
Globale Vuistregel Schattingen.....	17
Detail Schattingen obv Applicatieontwerp.....	19
Schattingen I/O.	20
Methoden en technieken.....	21
Ontwerp	21
Richtlijnen voor systeem inrichting	22
Voorspellings (prediction) modellen / technieken.....	24
Literatuur lijst.....	28

Inleiding

Dit hoofdstuk ... <reden van opstellen van dit document>

<veel termen in Engelse taal, om reden van ...> <maken we nog een onderscheid naar 'bouwende' en 'aanschaffende' partijen ?>

Leeswijzer, de structuur van dit document

De volgende hoofdstukken van het 'cookbook' zijn op de volgende IT-functies gericht:

1. CIO: 'Over de noodzaak van performance'
2. Technisch Architect: 'Typering van applicaties'
3. Systeem analist / ontwerper: 'Quantificering van workload'
4. IT Beheer Manager: 'Terugkoppeling productie - ontwerp'
5. IT Student: 'Methoden en Technieken'

Document historie

Datum	Auteur	Versie	Omschrijving
> 19-Jan-04	Jan Rauwerdink	0.1	Geen voorafgaand document
April 2004	Patrick Wolfs	0.2	Introduction
September 2004	Jan Rauwerdink	0.3	Reworked notes of committee meeting May 2004
Oktober 2004	Wim Luttkhuizen	0.4	Writing content on quantities, performance engineering, Archimate, CPU & elapstime assessments and methods and technics
December 2004	Laurens Blankers	1.0	Opmaak aangepast

De werkgroep

De NAF performance prediction werkgroep is als volgt samengesteld (alfabetisch op achternaam):

Naam	Organisatie
L. Blankers	TU-Eindhoven
H. Boer	<>
M. Chaudron	TU-Eindhoven
M. Jacob	Telematica Instituut
H. Jonkers	Telematica Instituut
H. Jurrien	Oracle
W. Luttkhuizen	BelastingDienst
J. Rauwerdink	Oracle
P. Wolfs	Sun Microsystems

Samenvatting

The Dutch (Netherlands) IT-Architects Forum [NAF] hosts several committees working on various architectural subjects. One of these subjects being performance of IT systems. To be more precise, *predictable* performance of IT systems. Hence, the 'NAF performance prediction committee' (see <http://www.naf.nl/performance/>), or 'the committee'.

The committee has decided to try and compile a 'how to' handbook, a 'cookbook' if you like, as a *practical* means of grasping the subject of application performance.

This 'cookbook' on performant application architecture and performance engineering aims at *raising awareness* that performance needs to be architected / engineered. We try to make our point by describing proven methods and highlighting common mistakes.

The committee realizes that '*an application*' is a very broad term; therefore architecting or engineering performance of 'an application' may be overly ambitious. A minimal distinction to be made is:

- 'Standard' applications (e.g. SAP, Oracle eBusiness Suite)
- 'Customized' applications (e.g. all custom build code)

You will find that these categories are used in the subsequent chapters of this document.

The committee finds that there are many publications on (predicting) application performance. Most of the information, however, is only useful to ICT (technical) specialists. Entry points for ICT decision makers are scarce. It is hard to find a method that works from a business need as a starting point towards a system engineered for performance.

In other words, ICT managers could overlook the need to keep performance (explicitly) within the scope of projects. IT architects may lack guidelines to engineer a system for performance.

This 'cookbook' attempts to explain performance from a perspective both the 'business' people and the 'employees' (specialists on the work floor) can relate to. All this without the need to be an ICT specialist. So, no discussions on network latencies or average storage device retrieval times.

You will find in this document focus on:

- 'Standard' applications; how to 'service' business processes with standard applications and identify the areas where attention on (predictable) performance is necessary.
- 'Customized' applications; which methods and tools are available, assuring development process efficiency with a 'guarantee' for (predictable) performance.
- The balance between costs and feasibility, 'best practices'. This is more technical focused and narrowed to specific technologies, which work more closely together with business logic on higher level.
- 'Data management' as the most important aspect of systems who determine performance. This will not mean that the other aspects are less important but reducing I/O will have the most effective impact on system behavior.

Aanpak overwegingen.

Aanleiding.

In de NAF werkgroep is besloten dat het werk van de Belastingdienst t.a.v. performance en het werk van het telematica instituut op dit gebied zal worden geïntegreerd in een kookboek. Hiervoor is veel tijd nodig in elk geval meer dan 1 dag per maand. Door de Belastingdienst is aan W.B.Luttikhuisen 1 dag per maand tijdsbesteding aan de NAF werkgroep toegemeten. Deze tijd is veel te kort om de gestelde NAF doelen, de productie van een kookboek, te realiseren. Gevraagd is om meer tijd welke vraag moet worden onderbouwd met een plan van aanpak met producten en stuur momenten van de NAF performance werkgroep. Mede gelet op het feit dat voor dit plan eveneens geen extra tijd wordt toegekend moet de onderbouwings vraag als weinig realistisch worden ervaren. Desondanks moet worden gepoogd om nog dit jaar de eerste versie van het kookboek te presenteren.

Aanpak.

Wij zullen in de projecten die lopen bij de Belastingdienst ons werk aan performance voortzetten. Omdat het project Archimate ook bij de Belastingdienst loopt zullen wij er voor zorgen dat er zoveel mogelijk samenwerking gaat ontstaan tussen de diverse complementaire projecten die bij de Belastingdienst op de gebieden architectuur en performance in uitvoering zijn. Wij zullen gebruik maken van de in deze projecten toegepaste termen, architectuur specificaties en documenten en proberen om de ITA producten en de Belastingdienst Performance Engineering producten en Archimate resultaten te combineren en te integreren. Geleidelijk aan zullen wij de resultaten trachten in te passen in de structuur van het NAF kookboek. Wij verwachten de eerste versie van het kookboek nog in 2004 te kunnen produceren.

Over de noodzaak van performance

Goede performance en voorspelbaar gedrag (snelle response, hoge throughput) van een applicatie, van een informatie systeem, wordt vaak als vanzelfsprekend gezien. Echter, zo goed als achter de draagkracht van een brug of de hoogte van een bedrijfsgebouw (ingewikkelde) berekeningen schuilen, zal goede performance en voorspelbaar systeemgedrag niet vanzelf ontstaan. Tijdens de ontwikkeling moet op kwantiteiten en efficiëntie gelet worden en voor de ingebruikname van een systeem dienen capaciteits berekeningen gemaakt te worden om goede performance te borgen. Bij performance spelen veel zaken een rol doch bij systemen die grote hoeveelheden gegevens verwerken is deze gegevensverwerking in de vorm van transactieverwerking in al haar facetten van doorslaggevend belang voor de prestaties van een systeem.

De technologie van transactieverwerking is vaak de sleutel tot het efficiënt, effectief en betrouwbaar verwerken van de dagelijkse stroom informatie op de geautomatiseerde middelen die wij tot onze beschikking hebben. Er moet worden bereikt dat gegevens en transactie servers een zo hoog mogelijke betrouwbaarheid krijgen en zo efficiënt mogelijk van de systeem resources gebruik maken. Transacties zijn de bouwstenen van de verscheidende informatiesystemen die algemeen worden geëxploiteerd. Transacties zorgen voor een consistente afhandeling van onze acties, en dragen er zorg voor dat parallelle verwerking kan plaatsvinden. Het ontwerpen en bouwen van transacties zodanig dat geoptimaliseerd voorspelbaar systeemgedrag ontstaat is daarom een wezenlijk onderdeel van het systeemontwikkelingsproces.

Een goed performend systeem stroomlijnt als het ware de bedrijfsprocessen, waar een system met slechte response de processen hindert en vaak (financieel) schade tot gevolg heeft. Voorspelbaar en geoptimaliseerd systeemgedrag kan niet, doch velen denken van wel, achteraf worden toegevoegd.

Modellen om tijdens de ontwikkeling van een systeem zicht te krijgen en te houden op het begrote resource gebruik en het capaciteit beslag alsmede de daarmee gepaard gaande kosten moeten in vroeg stadium worden ingezet.

Wellicht ten overvloede wordt opgemerkt dat het overwegend de response tijden en throughput (totaal aan verwerkte gegevens) zijn die de prestaties van een systeem bepalen. Het zijn deze aspecten die tijdens de realisatie van een systeem of delen daarvan veel aandacht moeten krijgen om tot goed ontworpen transacties te komen. Het gaat hierbij niet alleen om database en proces efficiëntie doch ook om eventueel netwerk verkeer. Ten aanzien van de response tijden kan worden gezegd dat snelheid wel een belangrijk punt is doch dat het zorgen voor regelmaat in het gedrag van het systeem zoals dat door de gebruiker wordt ervaren zeker zo zwaar weegt. Tijdens discussies over de mens machine interface (MMI) hoort men steeds meer beweren dat er tijdens de uitvoering van een taak er geen wachttijden mogen ontstaan doch dat er bij afsluiting van een taak best èèn of enkele seconden wachttijd acceptabel is.

Typering van applicaties

<voorstel ... uit [3]>

Many techniques have been developed for performance prediction of several classes of (concurrent) discrete-event systems, such as (parallel) computer systems (Ajmone Marsan et al. 1986; Gemund 1996; Jonkers 1995), telecommunication systems (Dowd and Gelenbe 1995), manufacturing systems (Yao 1994) and traffic. With the advent of the processor-oriented approach in business architectures (which are characterised by the increasing application of workflow management systems) it is apprehended that many logistics principles, and as a result, many techniques from the aforementioned fields are likely to be also applicable in the case of business architectures with little or no modifications.

<link maken van/naar typering>

Doel van applicatie typering (classificatie) is een 'kapstok' te maken waaraan inrichting- / tuning-aanbevelingen kunnen worden opgehangen. Ook zou classificatie een eerste ingang kunnen zijn om response van een (gegeven) applicatie te voorspellen.

Classificatie is noodzakelijk omdat naar doel van een applicatie de verwachtingen over response variëren. Bijvoorbeeld, van een interactieve web applicatie verwacht iedereen reactie binnen enkele seconden. Daarnaast wordt het heel normaal gevonden dat een salaris-'run' voor een groot bedrijf enkele uren in beslag neemt.

De getoonde tabel geeft een indeling (classificatie) van applicaties naar de *verwachting* van de reactie tijd (orde van grootte) van transacties binnen die applicatie. In de kolom 'Opmerkingen' wordt nader ingegaan op systeem karakteristieken.

Reactie tijd	Applicatie	Opmerkingen
msec	Real Time Processing (industriële autom. regelingen)	Buiten de scope van de werkgroep
sec	Interactief (OLTP) Client-Server over een LAN Producten: Oracle8i en Forms6i, DB2 etc. Type Client: Ultra Thin Client (terminal achtig)	CPU-bound op de centrale server, zelden een netwerk issue (100Mb, Gb). 'Klassieke' aanpak voor het tunen (en opzet) van een database: voldoende IO bandbreedte, CPU en memory, juiste init parameters en SQL tuning (i.e. beperken van IO).
sec	Interactief (OLTP) Client-Server over een WAN Producten: Oracle8i en Forms6i, SAP, BAAN, Peoplesoft, Siebel etc. Type applicaties: ERP, CRM Type Client: Heavy Clients, Thin Clients en Web Clients. Heavy Clients doen vaak stukjes OLTP op de Client CPU. Dit vraagt bandbreedte van het Netwerk	Vaak netwerk-bound, 100Mb - 56.6Kbps. De reactie tijd wordt bepaald door latency en throughput van netwerk, een (heel) snel WAN kost (veel) geld. Technisch is dit mogelijk, bedrijfs economisch meest niet, dan wordt veelal gekozen voor replicatie van data tot dichterbij de 'veeleisende' gebruikers-concentraties. Replicatie is echter ook duur, zowel in HW als in beheer (e.g. conflict resolution, consistentie). Nu er steeds meer bandbreedte goedkoper wordt, verdwijnt replicatie om redenen van reactie tijd (niet vanwege hoge beschikbaarheid) uit de belangstelling

Reactie tijd	Applicatie	Opmerkingen
sec	Interactief (OLTP) Internet - http Producten DB9i en 9iAS (OC4), SAP, BAAN, Peoplesoft, Siebel etc. Type applicaties: ERP, CRM, B2B, B2C Type Client: Heavy Clients, Thin Clients en Web Clients. Heavy Clients doen vaak stukjes OLTP op de Client CPU.	Netwerk-bound en afhankelijk van client CPU (igv Java). Vaak caching van statische en (deels) dynamische objecten op een (locale) tussenlaag om reactie tijd te beperken. Heeft deels de kenmerken van een C/S-WAN situatie, er kunnen tamelijk grote stukken code (Java classes) over de lijn komen. Die lokaal (c.q. lokaler) cachen beperkt de reactie tijd. Het is ook mogelijk inefficiënte SQL in Java in te pakken, daardoor neemt de load op de database onnodig toe. We zien wel dat de applicatie server als regel meer CPU en memory (nodig) heeft dan de database server.
sec	Interactief (OLTP) Internet - https Type applicaties: ERP, CRM, B2B, B2C Type Client: Thin Clients, Web Clients, Mobile Clients. HW SSL op routers, Software SSL op Client (kost CPU performance en bandbreedte)	Met name SSL opzetten (en actief houden) is een proces dat veel resource (CPU) kost. Vaak wordt hier HW voor ingezet ipv SW
Minuten	OLAP (OnLine Analytical Processing) , data analyse Type applicaties: BIW, BW (Grootte Databases met historische Data en veel rapportage transacties) Type Client: Thin Clients, Web Clients.	Veelal IO-bound -> maatregelen om IO te beperken, e.g. partitioneren, speciale indexen, aggregaties moeten reactie tijd beperkt houden
Uren	Stapel-gewijze (batch) processing Bijv. PL/SQL jobs, CICS	Afhankelijk van implementatie CPU- of IO-bound. Indien bedoelt dat een 'stapel' een set van interactieve transacties is, maar dan niet interactief uitgevoerd, dan gedraagt dit zich als OLTP, i.e. CPU bound. Doordat de gebruiker 'traagheid' weg is, worden (veel) kleine opdrachten heel snel achtereenvolgens aangeboden. Daar de 'normale' overhead van die transacties in veel kortere tijd moet worden weggewerkt, kan de gemiddelde reactie in deze vorm van 'stapelen' toenemen t.o.v. OLTP (wordt veel gevonden bij Object Oriented). In een andere 'stapel' vorm wordt de overhead maar eens in de zoveel transacties genomen (commit size). Doordat dan meer mutaties worden geschreven, raakt het proces als regel IO-bound. Goede IO spreiding (striping) kan tot (heel) hoge throughput leiden
Vele uren	Migraties, datawarehouses o.a. BIW en BW type applicaties die interfacen met ERP applicaties (data load)	Kenmerk van deze operaties is het verplaatsen van data van één kant van de storage naar een andere. Inherent zijn dit IO bound processen. Wanneer het IO systeem voldoende bandbreedte heeft, kunnen processen in parallel uitgevoerd worden, daartoe moet weer voldoende CPU capaciteit voorhanden zijn.

Kwantificering van workload

Kwantificering is de grondslag voor het ontwerpen van performance. Zonder deze grondslag is het niet mogelijk om voor een systeem performance te ontwerpen. Om van een dergelijk systeem voorspel gedrag te verwachten is dan irrieel. Ook bij het gebruik van SPE-ED en Archimate staan kwantitatieve analyses centraal.

Kwantiteiten

Hoofdzak: CRUD matrices maken, gebruikers groepen identificeren, frequenties bepalen

Ten einde de toekomstige workload van een applicatie in kaart te kunnen brengen dient de volgende vraag, hier zeer rechtlijnig geformuleerd, beantwoord te worden: "hoeveel gebruikers doen op welk moment wat met het systeem".

<en waarom wil je die workload kennen ? Vgl. Als je niet weet waar een gebouw voor gebruikt gaat worden kun je het niet ontwerpen, laat staan op verantwoorde wijze bouwen .. ditto een schip of een auto>

Iets nauwkeuriger: "met welke frequentie (intensiteit) gebruiken bekende types gebruikers het systeem op een zeker moment in de tijd ?" Enkele voorbeelden ter verduidelijking:

- Een tijdregistratie systeem; op vrijdagmiddag rond 16:30 vullen alle 1300 medewerkers hun tijdverantwoording over de afgelopen week in. Op maandagmorgen tussen 09:00 en 10:30 vragen alle 125 managers de 'inzet' rapporten voor hun respectievelijke teams op
- Een ERP systeem; iedere werkdag tussen 07:00 en 15:00 voeren 150 medewerkers orders in. Vanaf 15:00 start een batch die voorraden tegen orders controleert, bestellingen doet en voor facturering zorg draagt. Om 05:00 moet het systeem uit de lucht om een backup uit te kunnen voeren
- Een web site om theater kaarten te reserveren; iedere dag vanaf 09:00 to 22:00 kunnen klanten (aantal wisselt per seizoen, per dag, per evenement) reserveringen voor kaarten ingeven. Vanaf 22:00 worden opdrachten verzonden naar de aangesloten theaters (interface). Om 02:00 kunnen gegevens voor nieuwe voorstellingen worden opgestuurd door de aangesloten theaters (interface). Vanaf 05:00 worden alle transacties van de voorgaande dag gecopieerd naar het Data Warehouse voor marketing doeleinden

Wat uit de voorbeelden naar voren komt:

1. er zijn verschillen in soorten gebruikers van een applicatie
2. er zijn verschillen in typen transacties binnen een applicatie
3. er zijn verschillen in gebruik van een applicatie gedurende een tijdsinterval

Soorten gebruikers

Bij sizings en performance berekeningen (voorspellingen) wordt onderscheid gemaakt naar:

- 'named users'; gebruikers bekend in de applicatie, kunnen potentieel van de applicatie gebruik maken. Dit type gebruikers maakt geen aanspraak op (dure) systeem bronnen als CPU en memory.
- 'logged-on users'; gebruikers die een verbinding met de applicatie hebben gemaakt maar niet actief transacties uitvoeren. Dit type gebruikers maakt mogelijk aanspraak op memory (al dan niet swapped), maar gebruikt geen CPU
- 'active users'; gebruikers die actief transacties aan het uitvoeren zijn. Dit type gebruikers is het 'duurst', zij consumeren (real) memory en CPU.

Op een moment in de tijd kan een applicatie gekarakteriseerd worden door x 'named users', y 'logged-on users' en z 'active users', waarbij $x > y > z$. Om een goede performance prediction te kunnen maken moeten de grootheden x , y en z bekend zijn. Met name wanneer een nieuwe applicatie gepland wordt, kan het achterhalen van deze informatie (erg) moeilijk blijken hetgeen de kwaliteit van de voorspelling niet ten goede komt.

Verder wordt bij sizings en performance berekeningen onderscheid gemaakt naar:

- 'laag' actieve gebruikers
- 'gemiddeld' actieve gebruikers
- 'hoog' actieve gebruikers

Waarbij deze onderverdeling voor alle eerder genoemde klassen van gebruikers geldt. Dat wil dus zeggen dat op een moment in de tijd een applicatie gekarakteriseerd kan worden door y_1 'logged-on users, laag actief', y_2 'logged-on users, gemiddeld actief', y_3 'logged-on users, hoog actief', z_1 'active users, laag actief', z_2 'active users, gemiddeld actief' en z_3 'active users, hoog actief'. Dit detaillering niveau van de benodigde informatie maakt het achterhalen ervan vaak niet eenvoudig.

Soorten transacties

Wat als een transactie wordt ervaren is afhankelijk van het perspectief van waaruit wordt gekeken. Een gebruiker, een bouwer, en een beheerder hebben ieder een ander perspectief op wat zij als een transactie binnen een informatiesysteem ervaren.

Dit document hanteert de volgende algemene definitie voor een transactie:

een transactie is een verzameling operaties op de fysieke en abstracte toestand van een systeem.

Een transactie voldoet aan de volgende ACID (Atomicity, Consistency, Isolation, Durable) eigenschappen:

1. Atomicity.

De operaties van een transactie die de toestand veranderen zijn atomair, d.w.z. alle operaties van die transactie worden uitgevoerd of allemaal niet.

Dit moet gelden indien de transactie, de volledige applicatie, het operating system, of andere componenten normaal functioneren, maar ook in het geval van calamiteiten.

Een transactie is atomair indien het zich atomair gedraagt voor de 'waarnemer'.

2. Consistency.

Een transactie is een correcte overgang van de ene consistente toestand naar een andere consistente toestand. Dit betekent dat de resulterende toestand niet strijdig mag zijn met de integriteitsregels die gelden.

Een transactie produceert consistente resultaten, en indien dit niet mogelijk is, dan wordt de transactie afgebroken.

3. Isolation.

Isolatie betekent dat een transactie die alleen wordt uitgevoerd hetzelfde gedrag vertoont als in het geval dat gelijktijdige meerdere transacties worden uitgevoerd.

Het voldoen aan deze eigenschap worden bekeken vanuit het oogpunt van de 'waarnemer'. Dit betekent dat het intern in het informatiesysteem anders geregeld kan zijn.

4. Durability.

Een transactie is duurzaam als de resultaten na een succesvolle afronding duurzaam zijn vastgelegd. Anders bekeken, dit betekent dat als een transactie succesvol is voltooid dat dan de resultaten van de transactie weer terug te halen zijn in geval van een fout, storing of andere calamiteit.

Beschouw als voorbeeld een PIN-transactie bij een geldautomaat. Deze transactie is atomair als het geld wordt uitgekeerd EN de betreffende bankrekening wordt bijgewerkt. De transactie is consistent indien de integriteitsregel dat de hoeveelheid geld gelijk blijft niet wordt gebroken, d.w.z. als het bedrag dat wordt uitgekeerd gelijk is aan het bedrag dat van de rekening wordt afgeschreven. De transactie is geïsoleerd indien de transactie kan worden uitgevoerd terwijl ook andere transacties (bijv. het bijschrijven van het salaris) 'gelijktijdig' op de betreffende bankrekening kunnen worden uitgevoerd. En de transactie is duurzaam als na voltooiing van de transactie het saldo van de bankrekening de situatie weergeeft na de geldopname.

Een transactie is een verzameling operaties (of acties) op één database die logisch bij elkaar horen. De operaties, het wijzigingen of raadplegen van gegevens, moeten als één logisch geheel worden uitgevoerd.

Deze definitie volgt uit de relationele calculus welke ten grondslag ligt aan elke relationele database welke is gemodelleerd op basis van de in de wiskunde gedefinieerde verzamelingenleer (zie Bijlage C).

Om een duidelijke scheiding te maken tussen transacties die gegevens raadplegen en transacties die gegevens muteren worden de volgende twee soorten transacties onderscheiden:

- *Muteertransacties* die bepaalde gegevens wijzigen, inclusief het inbrengen en verwijderen van gegevens;
- *Raadpleegtransacties* die alleen gegevens bij elkaar halen om aan een bepaalde informatiebehoefte te voldoen.

Natuurlijk kunnen deze beide soorten ook in combinatie worden toegepast.

Tevens worden transactie onderscheiden naar middelenbeslag:

- *Gedefinieerde transacties*: transacties die gebruikmaken van een voorspelbare hoeveelheid middelen. Van deze transacties kan met zekerheid van te voren worden aangegeven wat de responstijden zijn.
- *Ongedefinieerde transacties*: transacties die gebruikmaken van een ongedefinieerde, en derhalve onvoorspelbare, hoeveelheid middelen.

Voorts kan van gedefinieerde transacties de volgende indeling nog worden gegeven:

- licht
- middel
- zwaar

System belasting profiel

<schema-tje van gebruik over een periode .. dag, week, kwartaal> <moet je voor de pieken ontwerpen of voor het continue gebruik, vgl. discussie over verbreding van snelwegen>

Bij het ontwerpen van performance gaat het om een keuze maken m.b.t. tot online taken en fabriekstaken. Meestal bestaan systemen uit deze componenten online en fabriek. Vaak wordt gekozen voor online taken omdat hier response tijden vaak bepalend zijn voor aspecten als gebruikersacceptatie. Bij fabriekstaken waar de systeem doorvoer (throughput) eigenschappen van belang zijn spelen aspecten als gebruikersacceptatie nauwelijks een rol.

Het ontwerpen voor pieken in een verwerking is in de regel niet gewenst. Indien voldoende aandacht wordt gegeven aan de transactie analyses en daarmee aan de schaalbaarheid aspecten van systemen kan "verbreding van de snelwegen" naar behoefte worden toegepast. Voorts kunnen pieken vaak worden voorkomen door goed te kijken naar de inrichting van bedrijfsprocessen.

Performance Engineering (PE)

<Hoofdzaak, meten in de praktijk van productie en terugkoppelen naar ontwerp>

PE betreft het o.b.v. de gestelde “Kwantitatieve eisen” t.a.v. performance en throughput bepalen van resourceverbruik en kosten aan de hand van de activiteiten “Capaciteitsschatting” en “Performance Metingen” (Benchmarking).

De schattingen worden gebruikt en op hun effecten doorgerekend in een Kwantitatief Analysemodel. Dit model is gebaseerd op het ArchiMate model zoals beschreven in het Archimate document “Quantitative Analysis of Enterprise Architectures”.

PE wordt uitgevoerd gedurende de gehele ontwikkelcyclus. Daarbij neemt de mate van detail en nauwkeurigheid van schattingen en metingen in de loop van het project toe en is uiteindelijk gebaseerd op actuele metingen:

1. Initieel, tijdens Definitiestudie, vinden schattingen globaal plaats o.b.v. referentieschattingen waarbij de applicatie wordt vergeleken met andere applicaties waarvoor het capaciteitsbeslag bekend is.
2. Tijdens Basisontwerp worden globale schattingen uitgevoerd o.b.v. vuistregels voor applicatietaken.
3. Tijdens Detailontwerp vinden gedetailleerdere schattingen plaats o.b.v. schattingen per ontwerpcomponent.
4. Tijdens Realisatie wordt het verbruik van de gerealiseerde applicaties (of onderdelen daarvan) gemeten en op basis daarvan een schatting voor productiesituatie gemaakt.

De resultaten van de activiteit “Capaciteitsschatting” worden vastgelegd in een “**Capaciteitsbeslag Rapport**” en de resultaten van de activiteit “Performance Metingen” worden vastgelegd in een “**Performance Rapportage**”.

Bepalend voor Performance en throughput is de **doorlooptijd** van de transacties welke bestaat uit de systeem verwerkingstijd evt. aangevuld met netwerk transporttijd. De doorlooptijd wordt bepaald door de capaciteit die een systeem heeft voor de verwerking van de transacties. Deze capaciteit wordt uitgedrukt in de hoeveelheid resources die aan de transacties ter beschikking gesteld kunnen worden. De kwantitatieve eisen die aan de applicatie en dus aan de transacties worden gesteld zijn daarmee bepalend voor de hoeveelheid resources die worden gevraagd en is daarmee een belangrijke kostenfactor.

De resources die van belang zijn voor performance en throughput zijn met name **CPU, I/O, Netwerk Bandbreedte** en **Schijf Opslag. Memory** is ook een belangrijk resource, maar is moeilijk te begroten en maakt relatief slechts een klein deel van de kosten uit. Dit resource verdient echter wel aandacht en wordt wel gemeten want gebrek aan dit resource heeft wel serieus impact op performance en throughput.

Onderstaand plaatje laat zien hoe de totale doorlooptijd is opgebouwd uit de componenten CPU (verwerkingstijd door de processor), I/O (verwerkingstijd voor het lezen/schrijven naar schijfopslag), Netwerk (transporttijd over netwerk) en uiteraard wachttijd (wachten op CPU, Netwerk, Database Locks etc.).



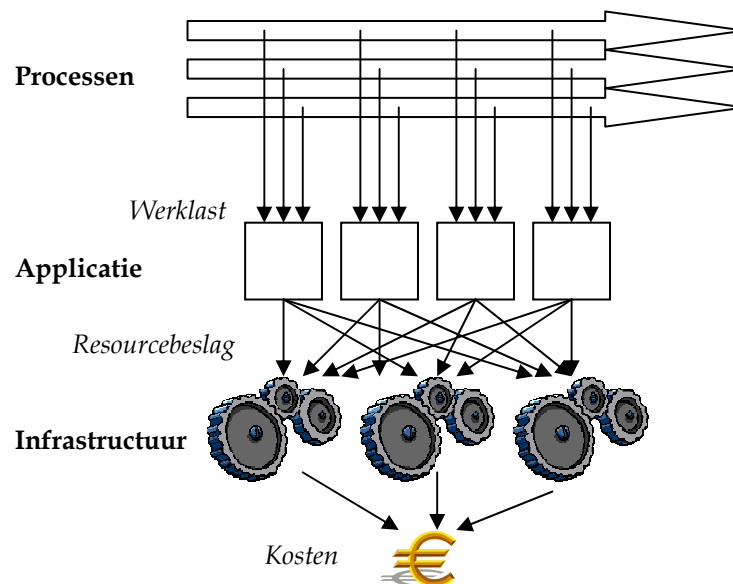
De CPU en Doorlooptijd gaan we tijdens definitiestudie en ontwerp schatten en tijdens realisatie (en productie) worden deze (en de detailcomponenten Wachtijd, CPU, I/O en

Netwerk) gemeten. De volgende paragrafen gaan verder in op verschillende werkwijzen en technieken voor het uitvoeren van schattingen en performance metingen.

Kwantitatief Analysemodel (ArchiMate)

Bij PE wordt gebruik gemaakt van ArchiMate, een onderzoeksproject van een aantal partijen: Belastingdienst, Telematica Instituut, ABN Amro, ABP, Centrum voor Wiskunde en Informatica, Radboud Universiteit Nijmegen en Ordina. ArchiMate definieert een hiërarchisch lagenmodel waarin op hoofdlijnen de niveaus “proces”, “applicatie” en “infrastructuur” worden onderkend. Dit model noemen we hier “Kwantitatief Analysemodel”, en biedt een kapstok om de kwantitatieve aspecten zoals performance en doorlooptijden op de genoemde niveaus in hun onderlinge verbanden door te kunnen rekenen.

Het principe van het ArchiMate Kwantitatief Analysemodel is dat de kwantitatieve eisen (zoals gebruiksfrequenties, doorlooptijden, responsetijden) die worden gedefinieerd op het bovenste proces niveau als eerste worden doorgerekend op de totale werklast die dat genereert op het tweede applicatie niveau. Vervolgens wordt de werklast op applicatie niveau verder doorgerekend naar derde infrastructuur niveau in termen van de door de applicatie geclaimde resourcebeslag (CPU, IO) en de daarmee samenhangende kosten. In onderstaande figuur wordt dit schematisch weergegeven.



In ArchiMate wordt onderscheid gemaakt tussen de verticale en horizontale berekeningen. Hierboven is het principe van een verticale berekening aangegeven van proces niveau, via applicatie tot infrastructuur niveau. Bij een horizontale berekening gaan we de keten in één laag doorrekenen. D.w.z. dat we voor een totale keten van bijv. processtappen op bovenste niveau het kritieke pad bepalen en de totale doorlooptijd van een verwerkingseenheid (zaak of stapel/batch, afhankelijk van het gedefinieerde proces) berekenen. Deze horizontale keten berekenen we op analoge wijze ook op het niveau van de applicatie waar we bijv. ketens van applicatieverwerkingen zoals bijv. transacties beschouwen.

Berekeningen kunnen we alleen maken als we inschattingen en aannames kunnen maken. Voor een horizontale berekening hebben we bijv. inschattingen nodig van doorlooptijden van processtappen en onderliggende applicatieverwerkingen. Voor een verticale berekening hebben we een inschatting nodig van de zwaarte van applicatieverwerkingen en het daarmee samenhangende resourceverbruik zoals CPU.

In de hierna volgende paragrafen geven we aan welke schattingstechnieken daarbij gehanteerd worden in de verschillende fasen van een project.

Verticale berekening

Als eerste verzamelen we o.b.v. de eisen de kwantitatieve aspecten op proces niveau. De volgende relaties en bijbehorende attributen willen we daarbij weten:

- **Relatie Proces-Applicatie:** Welke processen, deelprocessen dan wel procestaken gebruikmaken van welke applicatieverwerkingen. Als applicatieverwerking kunnen we bijv. een applicatietaak nemen conform BOA Functionele Constructie.
- **Gebruiksfrequenties:** De gemiddelde en maximale gebruiksfrequentie bij elke relatie tussen proces en applicatieverwerking
- **Relatie Applicatie-Infrastructuur:** Welke applicatieverwerkingen gebruikmaken van welke infrastructuur componenten. Als infrastructuur component kunnen we bijv. denken aan File Servers, Database Servers, Queue Servers, Applicatieservers (WebSphere WAS, CICS), Webservers etc..
- **Resourceverbruik:** In welke mate elke applicatieverwerking (bijv. applicatietaak) gebruikmaakt van resources in de infrastructuur componenten. Belangrijke resources hierbij zijn: hoeveelheid I/O, hoeveelheid CPU verbruik.

De rekenwijze is als volgt:

1. Sommeer per applicatieverwerking (bijv. Applicatietaak) de gebruiksfrequenties over alle relaties waarlangs de betreffende applicatieverwerking wordt benaderd door de verschillende (deel)processen/procestaken.
Voer de sommatie uit voor zowel gemiddelde als maximale gebruiksfrequenties. Aldus bepalen we dus de gemiddelde en maximale werklast per applicatieverwerking in termen van aantallen verwerkingseenheden per tijdseenheid (bijv. zaken/ dag, opvragingen/ uur etc.).
2. Sommeer per infrastructuurcomponent het product “werklast * resourceverbruik” over alle relaties waarlangs de betreffende infrastructuurcomponent wordt benaderd door de verschillende applicatieverwerkingen. Gebruik voor inschatting resourceverbruik de schattingstechnieken zoals verderop in dit document worden beschreven. Voer de sommatie uit voor zowel gemiddelde als maximale werklasten. Aldus bepalen we dus de gemiddelde en maximale resourceverbruik per infrastructuurcomponent in termen van CPU verbruik en aantallen I/O's.
3. Als we weten wat de beschikbare capaciteit is aan resources voor de infrastructuurcomponenten, bijv. als we weten hoeveel CPU capaciteit beschikbaar is, dan kunnen we berekenen wat de bezettingsgraad is van een infrastructuurcomponent. Indien blijkt dat er sprake is van een substantiële bezettingsgraad, dan zal hierdoor wachtrij vorming ontstaan. Dit moeten we ofwel voorkomen dan wel wachttijden calculeren of inschatten in de doorlooptijden van de betreffende applicatieverwerkingen.

Horizontale berekening

1. Maak een inschatting van de gemiddelde en maximale doorlooptijd van elke applicatieverwerking obv het ingeschatte resourceverbruik. Gebruik hiervoor schattingstechnieken zoals verderop in dit document worden beschreven. Indien sprake is van een substantiële bezettingsgraad van infrastructuur componenten moeten we wachttijden calculeren of inschatten in de doorlooptijden van de betreffende applicatieverwerkingen.
2. Maak een inschatting van de gemiddelde en maximale doorlooptijd van elke processtap obv het ingeschatte doorlooptijden van de onderliggende applicatieverwerkingen.
3. Voer in proceslaag en applicatielaag een kritieke pad analyse uit en bereken de doorlooptijden van de gehele keten op beide niveaus.
4. Controleer in de keten in hoeverre de doorlooptijden en responsetijden voldoen aan de gestelde eisen hieraan.

Schattingen CPU Verbruik & Doorlooptijd

CPU Verbruik wordt uitgedrukt in de hoeveelheid processortijd die een applicatie, of een onderdeel daarvan zoals een transactie, nodig heeft voor de verwerking. Dit kan uitgedrukt worden in bijv. "ms/transactie" of "ms/seconde". Het laatste getal wordt verkregen als de transactie throughput (transacties/sec.) wordt vermenigvuldigd met het verbruik per transactie en geeft aan hoeveel CPU tijd er per 'echte' seconde nodig is bij continue throughput aan transactieverwerking. Dit kan ook worden aangegeven in % wat aangeeft welk percentage van de CPU nodig is voor de applicatie of transactie.

Indien transacties alleen in bepaalde tijdsvensters worden uitgevoerd (bijv. alleen overdag), of sprake is van een gevarieerd throughput profiel per 24-uur venster, dan is het zinvol om de gemiddelde en maximale CPU tijd per dag te bepalen en de maximale CPU per uur of per sec. Het CPU profiel (verdeling CPU verbruik over de uren) is belangrijke input voor Exploitatie voor het opstellen van productieplannen zoals het "Master Production Schedule".

Voor kostcalculaties is het van belang om het gemiddeld CPU verbruik over langere periode, bijv. per jaar aan te geven. Het uitdrukken van CPU verbruik in ms heeft alleen betekenis als daarbij wordt verteld om welk processor het gaat en wat de verwerkingskracht daarvan is, bijv. uitgedrukt in MIPS en evt. wat de rekenkosten daarvan per tijdseenheid zijn.

Hierna worden een aantal werkwijzen voor CPU en doorlooptijd schattingen beschreven:

1. **Referentieschattingen** tijdens Definitiestudie waarbij de schattingen worden gebaseerd op een vergelijk met bestaande systemen met bekend (gemeten) resourceverbruik
2. **Globale Vuistregel Schattingen:** tijdens Basisontwerp waarbij de schattingen worden gebaseerd op kengetallen t.a.v. CPU en doorlooptijd voor "kleine", "gemiddelde" en "grote" transacties.
3. **Detail Schattingen obv Detailontwerp:** tijdens Detailontwerp waarbij schattingen worden gemaakt voor CPU en doorlooptijd per gegevenstoegang zoals in het detailontwerp gespecificeerd voor elke applicatietoegang. Dit wordt ook wel aangeduid als "Toegangspad Analyse (TPA)".

Referentieschattingen

De CPU Referentieschatting maakt gebruik van gegevens rond CPU verbruik die beschikbaar zijn (of te maken) van bestaande applicaties.

De nieuwe applicatie wordt dan ingeschat door een vergelijk te maken met deze bestaande applicaties, de verhoudingsfactoren tussen nieuwe en bestaande applicaties in te schatten en daarmee het CPU verbruik van de nieuwe applicatie te schatten.

De werkwijze is als volgt:

1. Identificeer een representatieve bestaande transactie. Representatief in de zin dat het platform (bijv. Unix, z/OS), de middleware (bijv. CICS, WebSphere), ontwikkeltools (C, Cobol, Java), database (DB2, Sybase) zo veel mogelijk overeenkomen.
2. Achterhaal wat het gemiddelde CPU verbruik " $CPU_{BESTAAND}$ " is van de bestaande transactie (in ms/sec of %).
3. Achterhaal wat de karakteristieken zijn van de bestaande transactie zoals hoeveelheid database IO en het type bewerkingen zoals: "basale input/output", veel regevaluatie, veel calculaties, veel stringbewerkingen etc.
4. Vergelijk de karakteristieken van elke nieuwe transactie "i" met de bestaande transactie en schat op basis hiervan een verhoudingsfactor " $V_{TRAN}(i)$ " voor het CPU verbruik van de nieuwe transacties t.o.v. de bestaande transactie.
5. Achterhaal de verhouding " V_{MIPS} " tussen CPU kracht van de bestaande en nieuwe transactie.

6. Achterhaal wat de gemiddelde throughput “TP_{BESTAAND}” (transacties/sec) is van de bestaande transactie.
7. Bepaal de throughput “TP_{NIEUW (i)}” (aantal transacties per tijdseenheid) van elke nieuwe transactie.
8. Bepaal het geschatte CPU verbruik “CPU_{NIEUW (i)}” van elke nieuwe transactie en tel deze op om het totale CPU verbruik “CPU_TOTAAL_{NIEUW}” te berekenen (in ms/sec of %).

$$CPU_{NIEUW (i)} = (TP_{NIEUW (i)} / TP_{BESTAAND}) * CPU_{BESTAAND} * V_{TRAN(i)} * V_{MIPS}$$

CPU_TOTAAL_{NIEUW}

CPU_TOTAAL_{NIEUW}

$$\sum_{i=1,n} = CPU_{NIEUW (i)}$$

Globale Vuistregel Schattingen

De globale capaciteitsschatting maakt gebruik van definities voor normtransacties “klein”, “middel”, “groot”. Daarbij worden vuistregels gebruikt voor het resourceverbruik en de doorlooptijd per normtransactie. Onderstaande tabellen geven een voorbeeld voor vuistregels voor mainframe applicaties tav doorlooptijd en CPU verbruik per transactiecategorie voor resp. fabriek en kantoortransacties. CPU is daarbij verdeeld in CPU verbruik door applicatieverwerking (in CICS) en verbruik door DB/2. De totaal CPU kan in dit voorbeeld worden gebruikt als normgetal voor de schattingen van het totaal gebruik door de applicatie.

Deze cijfers zijn gebaseerd op ABS metingen voor COOL:Gen fabriekstaken en (GUI) kantoor taken met een mainframe CPU van 230 MIPS (2064-1C3, zie bijlage A).

De bruikbaarheid van deze cijfers is uiteraard sterk afhankelijk van de actuele status van de gebruikte omgeving en dient regelmatig bijgewerkt te zijn. Een project kan dan ook het beste een eigen verzameling kengetallen hanteren die o.b.v. performance metingen regelmatig wordt geëvalueerd en bijgesteld. Deze kengetallen worden bijv. beheerd in het “Capaciteitsbeslag Rapport”.

Kantoor Normtransacties: CPU verbruik					

Per procestaak (of applicatietaak, afhankelijk van de te schatten input : functionele specificatie tijdens definitiestudie of functionele constructie tijdens basisontwerp) wordt een grove breakdown gemaakt in de verwachte technische transacties. Met technische transactie wordt bedoeld: een gegevensbenadering waarvoor het aantal SQL calls is in te schatten. Dit is geen detailontwerp maar een “kladmatige” schattingsaangelegenheid. Het aantal SQL

calls is een inschatting die de verantwoordelijke architect of ontwerper maakt en is een indicatie voor inschaling in de categorie klein/middel/groot. Op deze wijze kan per procestaak/applicatietask worden ingeschat wat het verwachte CPU gebruik is.

Bijv. voor de fabrieksmatige taken 1 en 2:

Taak 1 = 5* klein + 1* middel = 70 ms CPU / transactie

Taak 2 = 10* klein + 3* middel + 1 groot = 350ms CPU / transactie

Bij de schatting moet rekening worden gehouden met mogelijke opslagfactoren in CPU verbruik. Indien een applicatie boven de reguliere gegevensbenaderingen en gebruikersinteractie veel extra processing doet dan zal de hoeveelheid CPU uiteraard toenemen.

Potentiële opslagfactoren worden veroorzaakt door bijv:

- Complexe rekenfuncties
- Complexe regelgebaseerde evaluaties
- Veel stringmanipulaties (zoals in XML parsing)
- Gebruik van metagegevens (vergt extra I/O + CPU)

Bij het gebruik van de aangegeven kengetallen voor Fabriek en Kantoor normtransacties moet worden bedacht dat deze zijn gebaseerd op redelijk complexe transacties waarin bovengenoemde factoren reeds aan de orde zijn.

Om het totale CPU verbruik in te schatten zijn de Kwantitatieve eisen nodig, met name frequentie van events en de te verwerken systeeminputs. Hiermee kan, aan de hand van het procesmodel (en inzicht in de doorverwerking van frequenties van systeeminputs in de procestaak ketens) de transactierate per procestaak (applicatietask) worden ingeschat. De vermenigvuldiging van de transactierate met het CPU verbruik per transactie levert het aantal benodigde CPU seconden op, bijv. voor fabrieksmatige procestaken 1 en 2 met een transactierate van bijv. resp. 3 trans/sec en 1 trans/sec:

Taak 1 = 3 trans/sec * 70 ms/trans = 210 ms/sec.

Taak 2 = 1 trans/sec * 350 ms/trans = 350 ms/sec.

TOTAAL = 560 ms/sec = 0,56 sec/sec = 56 % CPU belasting.

Deze gegevens vormen voor Exploitatie belangrijke input voor het Capaciteitsplanning proces. Ook kan aan de hand van de openstellingstijden en de CPU kostprijsgegevens de CPU kosten voor de applicatie worden bepaald.

Naast resourceverbruik is het van belang om een controle te doen op de doorlooptijden van de fabriekstaken en responsetijden van kantoortaken. De vuistgetallen geven hiervoor de ervaringscijfers aan voor COOL:Gen/CICS/DB2 transacties. De doorlooptijden geven de mogelijke throughput aan die moet worden vergeleken met de hiervoor vastgestelde transactierates o.b.v. de Kwantitatieve Eisen. De responsetijden van kantoortaken moeten worden getoetst op haalbaarheid t.o.v. de gestelde eisen. Ervaring met COOL:Gen/CICS/DB2 kantoortaken is dat om een gebruikers responsetijd van < 1 sec. in 97% van de gevallen te bereiken hiervoor een gemiddelde CICS doorlooptijd nodig is van max. ca 280 ms. De relatief lage benodigde gemiddelde doorlooptijd wordt veroorzaakt door de additionele tijd benodigd voor de client/server verbinding en de relatief grote spreiding van de doorlooptijden. De client/server verbinding betreft een PPC verbinding tussen werkplek en mainframe. Werkplek naar PPC kost ca 10 % van de totale responsetijd. Afhandeling tussen PPC en CICS vergt ca 150 ms. Dit is een beperking van COOL:Gen generatie voor PPC en zal bijv. bij toepassing van TCP/IP minder zijn.

Detail Schattingen obv Applicatieontwerp

Als het ontwerp voor de functionele constructie is opgesteld (globaal in basisontwerp, dan wel gedetailleerd in detailontwerp), kan op basis hiervan een detailschatting van het resourceverbruik worden opgesteld. Dit wordt gedaan aan de hand van een schatting van het resourceverbruik en de doorlooptijd van de transacties. Deze schatting wordt binnen BCICT ook wel aangeduid met de term "Toegangspad Analyse" (TPA) en betreft een expertschatting die gebruikelijk door een DBA i.s.m. de (detail)ontwerpers wordt uitgevoerd. De hier beschreven schattingsmethode wordt ondersteund met een Excel tool (zie bijlage D: TPA Tool) waarin de schattingsberekeningen worden uitgevoerd.

Afhankelijk van het wenselijk detailniveau worden applicatietaken of gegevenstoegangen als transactie geschat. De schatting per transactie wordt gedaan aan de hand van CPU en doorlooptijd kengetallen voor het type SQL operatie dat wordt uitgevoerd. (Zie bijlage A en B voor verschillende kengetallen voor verschillende CPU en schijfopslag configuraties). Bij de schattingen is het tevens van belang om de kans in te schatten dat de te benaderen gegevens nog in cache (bijv. DB2 bufferpools) aanwezig zijn. Dit beperkt het aantal fysieke IO's (Get pages) en daarmee de doorlooptijd en (database) CPU. Deze kans wordt bij de schattingen uitgedrukt in het aantal verwachte fysieke pages dat wordt benaderd. (Een hulpmiddel "DB/2 Estimator" kan voor een specifieke SQL het aantal verwachte pages inschatten. Dit toegangspad kan in een actuele DB2 database ook worden berekend met de Explain functie). Hieronder een voorbeeld van een tabel voor toegangspad analyse:

Toegangspad Analyse							
Trans.	SQL Op.	Aantal	CPU [ms]	Doorloop Tijd [ms]	# Pages	# Indexen	Toelichting
T1	index	10	5	20	1	2	
		1	1	20	1	1	
	AL	11	6	40	2		
T2		5	2	10	2		
	Update	100	0,5	5	3		
	TOTAAL	105	2,5	15	5		

In deze tabel wordt per transactie (applicatietask of gegevenstoegang) 1 of meer SQL operaties gespecificeerd die uitgevoerd zullen worden. Per SQL operatie worden vervolgens schattingen gemaakt voor CPU verbruik en doorlooptijd aan de hand van kengetallen en een aantal parameters zoals het logisch aantal SQL calls, de hoeveelheid fysieke Page I/O's en het aantal 'geraakte' indexen. Per transactie worden de totalen opgeteld.

Een korte toelichting per kolom:

- **Transactie:** Naam van de applicatietask of gegevenstoegang.
- **SQL Operatie:** soort SQL operatie, bijv. "select op index", "insert", "bulkininsert".
- **Aantal:** Betreft het aantal logische SQL operaties. Voor 1-record select/updates/ insert operaties is dit het aantal keren dat de SQL wordt aangeroepen (en daarmee het aantal records). Voor meervoudige record cursor select operaties is dit het totaal aantal opgevraagde records evt vermenigvuldigd met aantal keren dat de SQL als geheel wordt aangeroepen.
- **CPU:** schatting van het CPU verbruik in aantal ms.
- **Doorlooptijd:** schatting van de doorlooptijd van de (enkelvoudige/niet parallele) SQL operatie in ms.
- **# Pages:** schatting van het aantal fysieke pages dat wordt benaderd. Dit betreft achtergrondinformatie waarop de doorlooptijd is gebaseerd.
- **# Indexen:** schatting van het aantal indexen dat wordt geraakt door de SQL operatie. Elke index draagt nl. bij tot het aantal CPU sec en het aantal Page I/O's.

- **Toelichting:** eventueel aanvullende uitleg als achtergrond bij de schatting.

Als de verschillende TPA schattingen voor alle applicatietaken van de applicatie zijn uitgevoerd, wordt een totaalschatting opgesteld waarin het totale CPU verbruik en de verwachte throughput per applicatietask wordt berekend.

Om het totale CPU verbruik in te schatten zijn weer (net als bij de globale schatting in de paragraaf hiervoor) de Kwantitatieve eisen nodig waarbij de vermenigvuldiging van de transactierate met het CPU verbruik per transactie het aantal benodigde CPU seconden oplevert, bijv. voor fabrieksmatige procestaak 1 met een transactierate van bijv. resp. 3 trans/sec :

$$\text{Taak 1} = 3 \text{ trans/sec} * 45 \text{ ms/trans} = 135 \text{ ms/sec.}$$

De doorlooptijd van Taak 1 laat hierbij een max. throughput zien van 4 transacties/sec. en moet weer gecontroleerd worden op de gestelde kwantitatieve eisen.

De doorlooptijd van een enkelvoudige (d.w.z. niet concurrent uitgevoerde) transactie wordt i.h.a. bepaald door de I/O wachttijd, tenzij er wachttijd optreedt a.g.v. bijv. CPU gebrek. Kengetallen voor deze I/O wachttijd worden bepaald door de specifieke database omgeving en de specificaties van de disksystemen. Bijv. een traditioneel Mainframe DASD 3390 systeem heeft ca 20 ms per page nodig voor random access en ca 2 ms per page voor sequential access. De huidige (2003) B/CICT mainframe configuratie is sneller, ca 4 - 6 ms random access (en ca 0.5 ms voor sequential access?). Gegevens kunnen echter in cache zitten waardoor de tijden nog veel lager worden, afhankelijk van het soort cache. Bijv. 3390 controller cache: 1-4 ms, global bufferpool: 0,2 ms. Of gegevens in cache zitten hangt af van de dimensionering en configuratie van de bufferpools en het soort gebruik dat hiervan wordt gemaakt. Een applicatie die exclusief gebruikt maakt van bufferpools en deze aaneengesloten benadert heeft een veel hogere buffer hit kans dan bufferpools die worden gedeeld door verschillende concurrerende applicaties. De ervaring met ABS applicaties is dat in een eigen partitie, zonder concurrerende applicaties, gemiddelde I/O tijd ca 1 ms kan bedragen. In productie kan dit veel lager zijn, bijv. tot ca 8 ms.

Uiteindelijke tijden voor schrijf en leesoperaties voor een SQL operatie zijn afhankelijk van het toegangspad, de hoeveelheid te benaderen pages (Fysieke I/O's) en de kans op buffer hits. Bijlage B geeft een voorbeeld van kengetallen voor CPU en doorlooptijd voor verschillende type gegevensbenaderingen. Ook voor deze kengetallen geldt dat een project het beste een eigen verzameling kengetallen kan opstellen en deze obv performance meetresultaten evalueren en bijstellen. Deze kengetallen worden bijv. beheerd in het "Capaciteitsbeslag Rapport".

Schattingen I/O.

I/O betreft de hoeveelheid lees/schrijf operaties die door een applicatie of transactie per tijdseenheid naar schijfopslag worden gedaan. Met name voor batchverwerking was dit in het verleden één van de meest kritische performance bepalende resources. Afhankelijk van de hoeveelheid CPU en I/O capaciteit is een transactie CPU-bound of I/O-bound. D.w.z. dat de voornaamste wachttijd en bijgevolg doorlooptijd door CPU gebrek, dan wel gebrek aan I/O capaciteit wordt bepaald. Tegenwoordig zijn transacties meestal CPU bound, tenzij uiteraard sprake is van grote hoeveelheden SQL calls, met elk grote hoeveelheid aan tabellen en grote hoeveelheden record aantallen per tabel. Deze hotspot transacties worden met een ITA analyse geïdentificeerd. De ITA analyse met de schattingen van aantallen SQL calls en van de aanroep frequenties vormt dan ook het uitgangspunt voor een I/O schatting. Het gaat hier om de vermenigvuldiging van de transactierates met de hoeveelheid geschatte I/O calls. Op deze manier kan worden vastgesteld hoeveel I/O's de applicatie in totaal per tijdseenheid zal uitvoeren. Ook hier is het van belang aan te geven of er sprake is van een verdeling in de tijd (bijv. per seizoen, maand, dag, uur etc.) en zoja hoe deze verdeling eruit ziet. Exploitatie voorbereiding kan hiermee rekening houden bij het plannen van de capaciteit over alle te draaien ICT services.

Er moet ook worden gekeken naar de kritische gevallen uit de ITA analyse en worden vastgesteld in hoeverre hier mogelijk wordt aangelopen tegen de grenzen van de bandbreedte van de I/O kanalen naar de opslagdevices ("DASD's" op het mainframe).

Ontwerp

Performance moet net als functionaliteit worden ontworpen. Dit gebeurt dan ook op dezelfde wijze, eigenlijk analoog daaraan, waarop functionaliteit wordt ontworpen. Bij het ontwerpen van performance is het niet zo dat men daar hele slimme mensen voor nodig heeft. Het helpt natuurlijk wel als deze mensen voldoende ervaring hebben in het ontwerpen van systemen en met name de performance. Belangrijker is echter dat er beseft wordt dat aan de basis van het ontwerpproces, kwantiteiten liggen die bij elke afweging moeten worden betrokken. In het ontwerpproces kan men verschillende stappen en fases onderscheiden. Een en ander begint met het **begrijpen** van het probleem, **creëren** van een oplossing, **beschrijven** van een mogelijk product of een oplossing en het **evalueren** van de geschiktheid daarvan. Het evalueren moet bestaan uit het **schatten van de correctheid**, het **schatten van de haalbaarheid** en het **afwegen van een eventuele voorkeur** t.o.v. andere oplossingen.

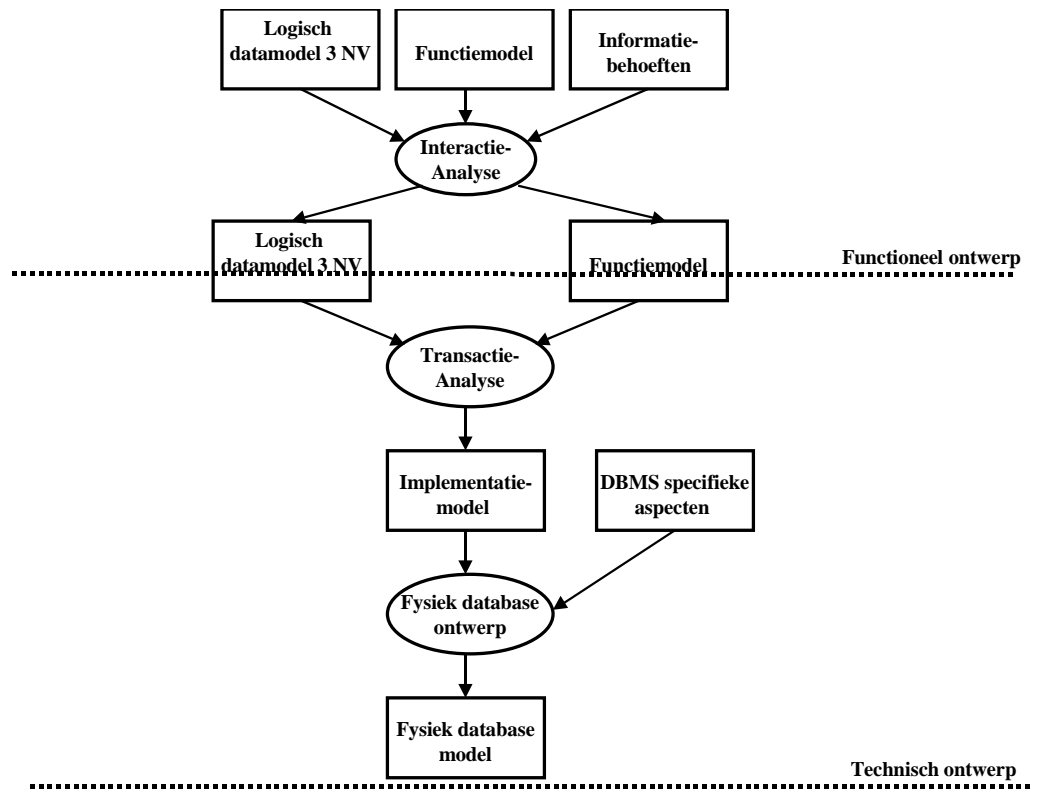
Als dit allemaal gedaan is mogen de volgende voordelen verwacht worden:

- Verhoogde productiviteit van ontwerpers omdat er geen tijd meer verkwist hoeft te worden met het technisch herontwerpen van een systeem
- Verhoogde kwaliteit en bruikbaarheid van het systeem of service door selectie van bruikbare ontwikkel en implementatie alternatieven
- Beheersing van de kosten van de ondersteunende hardware en middleware door indentificatie van wat nodig is en het verkrijgen van voldoende tijd om de juiste zaken bij/van leveranciers te vergelijken
- Verbeterde productiviteit tijdens testen en implementatie door ervoor te zorgen dat er voldoende resources tijdig beschikbaar komen

Het is niet mogelijk om performance problemen re-actief op te lossen. Het aanbrengen van performance nadat gebleken is dat er performance problemen zijn zal leiden tot sub-optimalisatie. En bij sub-optimalisaties blijven er altijd problemen achter. Het is noodzakelijk om pro-actief om te gaan met performance en deze te ontwerpen.

Bij het ontwerpen van performance wordt in eerste instantie het 'Data Management' gebied in beschouwing genomen. In de praktijk is gebleken dat zich hier de meeste problemen voordoen. Als men in CPU tijden spreekt van 1 micro seconde voor de uitvoering van een instructie en in I/O tijden van 30 miliseconden voor het transport van een fysieke page heeft men als mens geen goed beeld van de verhoudingen tussen deze getallen en de waarde die zij moeten voorstellen. Als men echter, deze brengt in een meer menselijke maat, dan zal als de micro seconde 1 seconde is de 30 mili seconden meer dan 8,5 uur zijn.

Dit geeft aan dat het beperken van de I/O en de veelheid van knelpunten die in de interactie tussen processen en gegevens voor kunnen komen een groot deel van response en throughput wensen op haalbaarheid kunnen worden beoordeeld. Voorts zal er door oplossing van knelpunten en optimalisaties een voorspelbaar systeem gedrag ontstaan.



Richtlijnen voor systeem inrichting

In grote lijnen kunnen bij de onderscheiden applicatie vormen (classificatie) richtlijnen voor de inrichting van een platform worden gegeven, vanuit het oogpunt van reactie tijd (response, performance):

1. Interactief (OLTP) Client-Server over een LAN;

De inrichting van de database server heeft de meeste aandacht, hier moet voldoende memory in zitten om de active client processen altijd in real memory te hebben (naast een SGA), voldoende CPU (> 1) om concurrent requests af te kunnen handelen, maar ook afdoende IO bandbreedte om mutaties snel door te kunnen voeren. 'Traditioneel' een grote Unix SMP machine, inrichting parameters zijn:

- #concurrent (actieve) gebruikers
- verwachte MBs mutaties/tijd (schrijven, lezen is 99% cache)

Tuning parameters (maatregelen) zijn:

- cache efficiency verhogen (memory parameters)
- SQL statement doorlooptijd verlagen door bijv. Indexering
- IO wachttijden verlagen door bijv. striping van files en/of toevoegen van devices

2. Interactief (OLTP) Internet - http; Bijv. DB9i en 9iAS (OC4J)

Dit is een twee-tier en drie-tier platform, een gescheiden database- en applicatie-server en presentatie-server. Gescheiden omdat in de praktijk o.a. Java processen ongelimiteerd CPU en memory *kunnen* aanspreken. Wanneer dit zou gebeuren naast de database processen, neemt de response hiervan snel af.

Nu veel business rules in Java worden gecontroleerd, verschuift CPU behoefte van de database- naar de applicatie-tier. In beginsel heeft de applicatie-tier nauwelijks IO-behoefte (behalve als bijv. Apache logging aan staat), maar des te meer memory en CPU. De database-tier heeft, als vanouds, wel IO bandbreedte nodig om mutaties snel te verwerken. Inrichting parameters zijn:

- #concurrent (actieve) gebruikers -> applicatie-tier
- verwachte MBs mutaties/tijd (schrijven, lezen is 99% cache) -> database-tier

Tuning maatregelen zijn:

- Java VMs tunen (memory, connection pool)
- SQL statement doorlooptijd verlagen door bijv. Indexering
- IO wachttijden verlagen door bijv. striping van files en/of toevoegen van devices

Stapel-gewijze (batch) processing; Bijv. PL/SQL jobs, CICS

In de praktijk blijkt dat zgn. Object Oriented programming in een batch verwerking niet erg goed werkt. Een batch is in zo'n omgeving (inderdaad) een stapeling van individuele opdrachten die op dezelfde wijze wordt uitgewerkt als ware het n OLTP opdrachten. Hier is echter geen efficiency voordeel mee te halen, de (meest) kleine transacties die frequent gecommited worden brengen veel overhead met zich mee en deze moet in kortere tijd worden weggewerkt dan wanneer dit bij daadwerkelijke OLTP (gebruiker wachttijden !) het geval zou zijn. Het gevolg is veelal een *toename* van de gemiddelde reactietijd.

De 'klassieke' batch processing werkt met PL/SQL jobs (procedural SQL), te verwerken records worden in een cursor geselecteerd, soms gesplitst in parallelle jobs (a-d, e-i, j-r, s-z) en met speciale code verwerkt die soms een array als in/out-put heeft. Transacties worden per grote groep (25.000 of 100.000) tegelijk gecommited, daarbij 'stokt' het proces vaak omdat veel IO geschreven moet worden.

Inrichting parameters zijn:

- #parallele processen
- verwachte MBs mutaties/tijd (schrijven, lezen is 99% cache)

Tuning maatregelen zijn:

- memory 'hot spots' vermijden (latches)
- SQL statement doorlooptijd verlagen door bijv. Indexering
- IO wachttijden verlagen door bijv. striping van files en/of toevoegen van devices

Voorspellings (prediction) modellen / technieken

<uit [3]>

Quantitative enterprise architecture aspects and modelling

Uit Shallahamer [2]: "...

Forecast Model	Scope	Required Data	System Status	Precision Possibility	Project Duration
Simple Math	component	application specific	production proposed	low	very short
Ratio Modeling	component	detail application and broad system	production proposed	low	short
Linear Regression	component	detail application and broad system	production	high	medium
Simple Queuing	component	transaction detail	production proposed	high	long
Simulation	component system	detail application and broad system	production	high	medium

Figure 1. This table can be used to determine an appropriate forecast model. Ask yourself about the project scope, forecast model required data, system status, precision possibility and precision required, and how much time you have. Asking and answering these questions will lead you to your best forecast model options. Keep in mind that typically multiple forecasting models are appropriate."

Uit Shallahamer [2]: "...

Simple Math. As the name states, it's *simple*. The funny thing is, we use simple mathematical forecasting all the time without ever thinking about it. For example, if we know an Oracle client process consumes 10MB of non-shared resident memory and plans are to add another 50 users (resulting in 50 additional client processes),¹ then the system will require 500MB of additional memory. Because there is no queuing involved with *Simple Math*, it's only appropriate for forecasts like memory, basic IO predictions, and basic networking. I would resist using simple math for CPU, advanced IO, and advanced network predictions.

Ratio Modeling. Ratio modeling is a fast way to make low precision forecasts. Myself and two colleagues developed the *Ratio Modeling Technique* back in the mid-1990's in response to finding ourselves in situations where we simply had to deliver a forecast, yet we did not have the time to use an alternative model. *Ratio Modeling* works very well when you are quickly budgeting hardware, assessing and exposing technical risk, validating alternative technical architecture designs, and especially

¹ We all know that adding fifty more users does not equal fifty more server processes. This is when understand the concepts of named users, connected, and active users becomes important.

when *sizing packaged applications*. The technique enables you to define the relationship between process categories (e.g., batch processes) and a specific system resource (e.g., CPU). *Ratio Modeling* produces no statistical data, so it is truly a *back of the envelop* forecast technique. It has been so useful, there is a separate paper devoted entirely to understanding and using [Ratio Modeling](#).

Linear Regression (LR). Linear regression is fantastic because it's simple to use (assuming you have a [good tool](#)²) and provides statistical measurements to add strength to your forecast statements. I typically use *LR* when I want to quickly predict CPU utilization for a given business activity like *orders shipped per hour* or *web hits per second*. Regression analysis requires data and data requires a system to sample from. Therefore, *LR* requires a production environment.

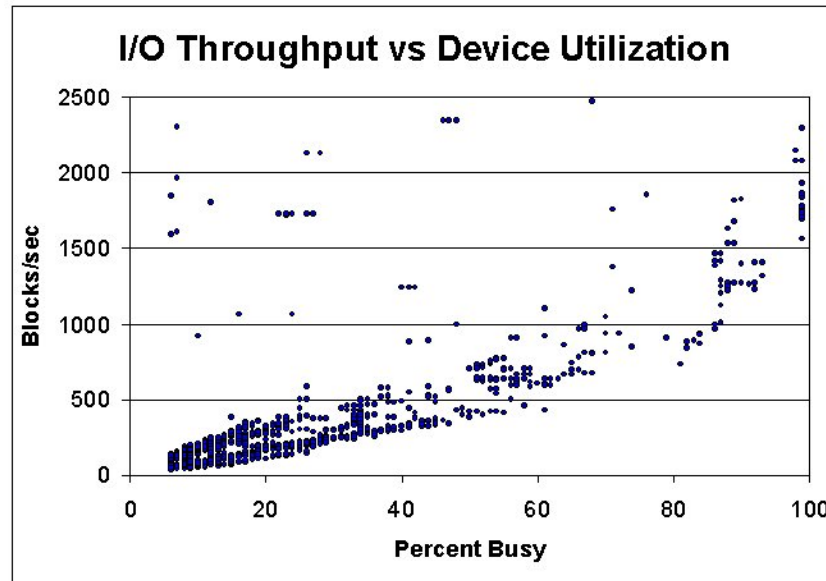


Figure 2. This is the graphical chart from OraPub's simple (single independent value) linear regression workbook. If you carefully draw vertical line at around 75% busy, you will notice to the left of the line the data is fairly linear, but to the right, the data is increasingly non-linear. This is why one should never use linear regression to forecast past 75% utilization.

The relationship between just about anything and CPU utilization is *not* linear. Forecast techniques and reality demonstrate that at around 55% utilization, queue time is already consuming around 5% of total response time. And at around 75% utilization, queue time is already consuming around 15% of total response time. So while I enthusiastically use linear regression, I will never make a forecast with a utilization over 75%. Simply because even though the forecast model says it's OK, I know it's not because it expects the data to be linear when it's clearly not. So be careful when using *LR*, but use it liberally and appropriately.

Simple Queuing. Queuing theory is a wonderful and powerful forecast technique. It can even be implemented into an [MS-Excel workbook](#).³ Understanding queuing is a fundamental performance management concept. The components are simple: a transaction, a queue (or *line* as we say in the USA) and a server. When a transaction enters a queue response time starts and queue time starts. When the transaction

² You will find free MS-Excel based linear regression tools that are very simple to you on OraPub's web-site. While simple, they are very powerful and useful.

³ Again, you'll find a very nice free an MS-Excel based queuing theory workbook on OraPub's web-site.

finally begins getting served by the server (a CPU perhaps), queue time stops and service time begins. When the transaction is finished being served, service time stops and response time stops. This can be symbolically⁴ shown mathematically:

$$R_t = S_t + Q_t$$

Based upon basic queuing theory and your personal experience, you should be able to *feel* that the below graph is correct.

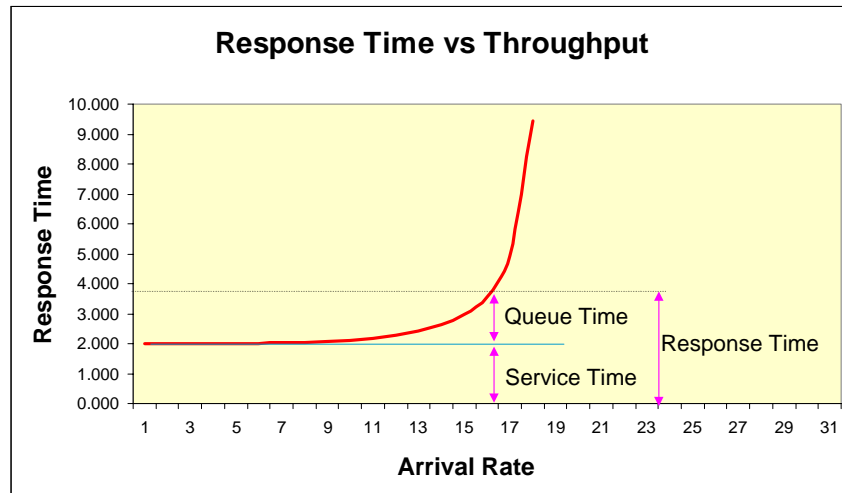


Figure 3. This is a classic response time curve taken from OraPub's MS-Excel based queuing theory workbook. Notice that for quite a while, response time equals service time. But with an arrival rate of around 10 trx/sec, queuing starts to enter the system. This occurs at around 55% utilization. By the time utilization is around 75%, queue time accounts for around 15% of response time and after that it's exponential. This is why a system may have an acceptable response time, but then *all of a sudden* performance dramatically degrades.

With a solid basic understanding queuing theory, both reactive and proactive performance management will take on a new meaning. This is why I talk about response time and its components in my [Reactive Performance Management](#) course and spend half a day on basic queuing theory in my [Proactive Performance Management](#) class.

When more than one queue is involved, a network or circuit queuing model is born. For Oracle based *systems*, I have found the only way to get respectable predictions is by using a multiple queue network. While a single queue model does model the CPU subsystem rather precisely, it can *not* model the entire system and the affect one subsystem (e.g., CPU) has on the other subsystems very well. Plus, for detailed forecasts, actual transaction data must be gathered, organized, and entered into the queuing model. This can take a relatively long time. So long in fact, that many times is less expensive simply to purchase more hardware instead of paying for a transaction based queuing theory forecast study.

⁴ Symbols can be powerful models. Mathematics is only one of the many symbolic languages. Symbols are powerful because they represent much more than they are. This is one reason why symbolism is so important to us as people.

Simple queuing theory forecasting models are also excellent learning tools. For example, it is very easy to visibly see the results of tuning, adding *additional* CPUs, or using *faster* CPUs.

Simulation. While common amongst the world's research community, simulation is virtually ignored when forecasting Oracle system performance. Cost and education are the primary reasons, but with some instruction, a little experience, along with some basic statistics, simulation can be a fantastic solution that yields surprisingly good precision.

Many commercial simulation packages are graphical and can produce some eye-popping "ah ha" moments when presenting the animated results to management. However, these commercial packages can be relatively expensive and have a substantial learning curve. But there are alternatives as I will discuss below.

Basically, a simulation forecast model simulates transactions working their way through a system (in our case, an Oracle based system), while watching and recording what is happening. For usability and precision⁵ reasons, the model is highly abstracted with the entire computing system represented with only a few queues. To enable the simulation to mimic your production system, it must have the ability to be *tuned*. The tuning process can take a long time if done by hand."

⁵ It may seem odd, but it is nearly always the case that the more detailed or complex a forecast model becomes, the less precise and usable it becomes. Take for example, linear regression. Adding more and more independent variables nearly always reduces the forecast precision. Don't be fooled into thinking that detailed forecast models are "better predictors"...because they are not.

Literatuur lijst

1. Morle, J.; *Scaling Oracle8i – Building Highly Scalable OLTP System Architecture*, Addison Wesley, Dec 1999, ISBN 0-201-32574-8
2. Shallahamer, C.; *Responsibly Forecasting Oracle System Performance*, OraPub Inc., May 2003, <http://www.orapub.com>
3. Iacob, M-E. & Jonkers, H.; *Quantitative Analysis of Enterprise Architectures*, Telematica Institute, Dec 2003, TI/RS/2003/093
4. Connie M Smidt Software Performance Engineering (SPE)