HZD
Hessische Zentrale für Datenverarbeitung

HESSEN

Mehr Performance ohne Hardware-Upgrade

Road Map zur Identifizierung und Beseitigung der Hauptursachen langer Antwortzeiten

Dr. Martin Rosenbauer
(MTI)

Boris Noack,
Ulrich Rompel
(HZD)

HZD
Hessische Zentrale für Datenverarbeitung

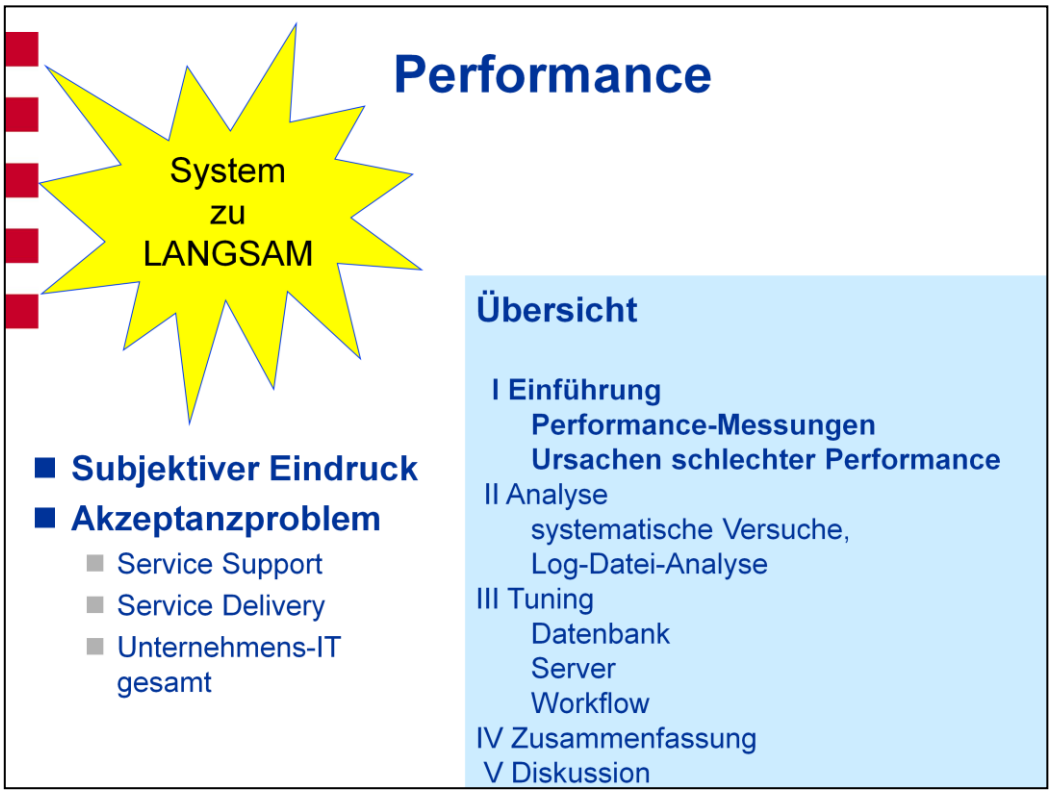
- Rechenzentrum der Hessischen Landesverwaltung
- Lösungen für die Verwaltung
- Wiesbaden und Hünfeld
- 600 Mitarbeiter
- Umsatz 130 Millionen Euro

MTI Management of Technology and Information Dr. Martin Rosenbauer

Workflow-Projekte - ARS-Programmierung

- Projektleitung für AR System Projekte
- Beratung bei Workflow-Konzeption
- Umsetzung von Workflows
- zeitlich befristete Verstärkung von ARS-Teams
- Schwerpunkte:
  AR System, Perl, Oracle, SQL, UNIX, Schnittstellen

2

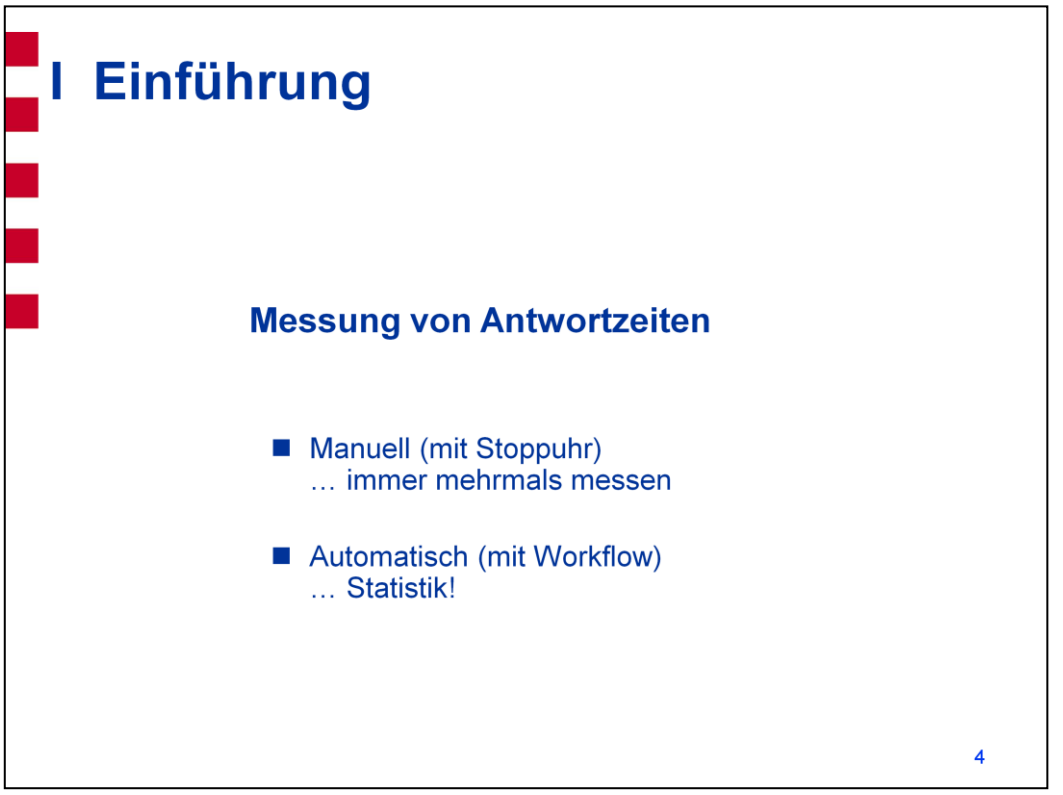


„System ist langsamer als gestern…“, „ist unglaublich träge“, …

- Vortrag gibt kein „Patentrezept“
- Tuning ist Prozess, nicht „*einmalige Maßnahme*“

Teil I des Vortrags: Einführung

1. Schritt: Quantifizierung des Begriffs „zu langsam“
2. Überblick über Architektur und Bottle-Necks



… mit Bleistift und Papier

… oder mit Active Links (z.B. On Submit/Modify mit Execution Level 0 und „on after submit“/“after modify“

Auswahl nach versch. Gesichtspunkten:

•Welche Aktionen sind besonders störend für die Benutzer ?

•Welche Aktionen laufen weitgehend/ausschließlich lokal im Client / Browser (Aktive Links, GUI-Operationen etc.)

•Welche Aktionen verlaufen ohne Einfluss des Clients (z.B. Datenbankabfragen, Aufbau von Tabellenfeldern etc.)

•Keine zu kurzen Aktionen messen (>1,5s), sonst zu große Messungenauigkeit/Varianz

Unser Referenz-Scenario

Aktion	Dauer* (s)
Aufruf „e-inventory“-Maske	18,8
Wechsel Bearbeitungsmodus	7,5
Wechsel Anzeigemodus	0,8
Erweiterte Suche aufrufen	4,4
Erweiterte Suche abbrechen	1,7
Erweiterte Suche, Suche starten	1,8
Speichern im Bearbeitungsmodus	11,8
Fokus setzen im Bearbeitungsmod	4,9
Eingaben löschen in Schnellsuche	10,1
Umzugdialog aufrufen	8,2
Umzug abbrechen	1,6
Umzug durchführen	0,9

Aktion	Dauer* (s)
Umzugsdialog schließen	2,1
Aktivierung aufrufen	5,6
Aktivierung durchführen	1,5
Aktivierungsdialog schließen	1,7
Austausch aufrufen	9,7
Austausch abbrechen	2
Austausch durchführen	2,6
Austausch schließen	1,1
Ersterfassung aufrufen	5,7
Ersterfassung abbrechen	1,4
Berichte aufrufen	1,2
Summe	**107,1**

* Vor der Performance-Optimierung

5

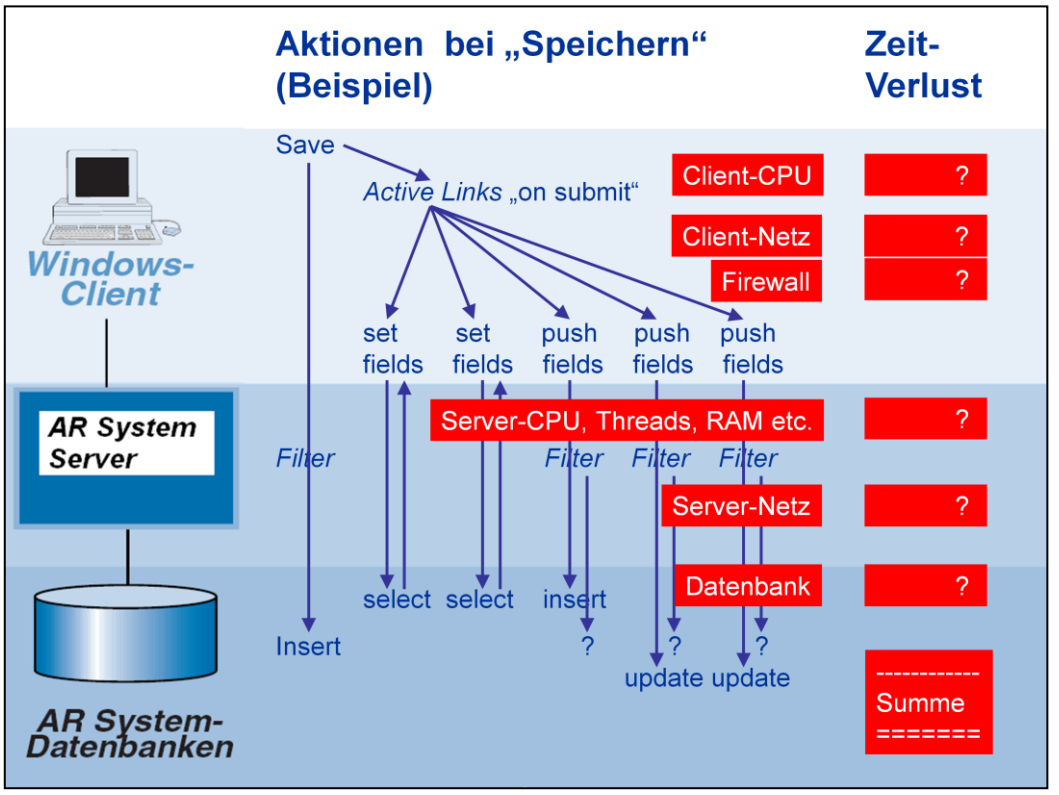
Scenario entspricht normalem Arbeitsvorgang der meisten Kunden

Der Einfachheit halber von Hand mit der Stoppuhr gemessen.

Durschnittsdauern aus jeweils mehreren Messungen (mindestens jeweils 3 Messungen). Reproduzierbarkeit prüfen!

Cache-Problematik ! (User Tool)

… woher kommen die langen Antwortzeiten?



<Architektur erklären>

Einführung für Nichtexperten – Wie funktioniert das Action Request System?

Sowohl im Client wird auf Vorgänge reagiert (Active Links) als auch im Server (Filter).

Willkürliches Beispiel: Submit-Vorgang eines Trouble-Tickets. Set-Fields:

- Nachschauen ob Telefonnr und Adresse schon vorhanden und mit Benutzer-Eingaben übereinstimmen (Anfrage von UT an Server, dann Anfrage von Server an Datenbank,
- dann Antwort der Datenbank an den Server, dann Antwort des Servers ans UT, dass beides vorhanden, aber falsch),
- Speichern eines Statistik- oder History-Datensatzes (Filter!). (Insert)
- Submit (Insert) des neuen Trouble Tickets (Filter!).
- Update von Adresse (Filter!)
- Update von Telefonnummer (Filter!)

Jede Aktion (ob Öffnen eines Menüs, Wechsel angezeigter Registerkarten, Öffnen von Dialogform…) ist verschieden.

< Vorteil von Filtern>

Bei langen Antwortzeiten:

- Zunächst nur die „Summe“ bekannt

→ Fischen im Trüben (Viele Parameter)

Prinzipiell 2 Möglichkeiten der Ursachenfindung:

Übersicht

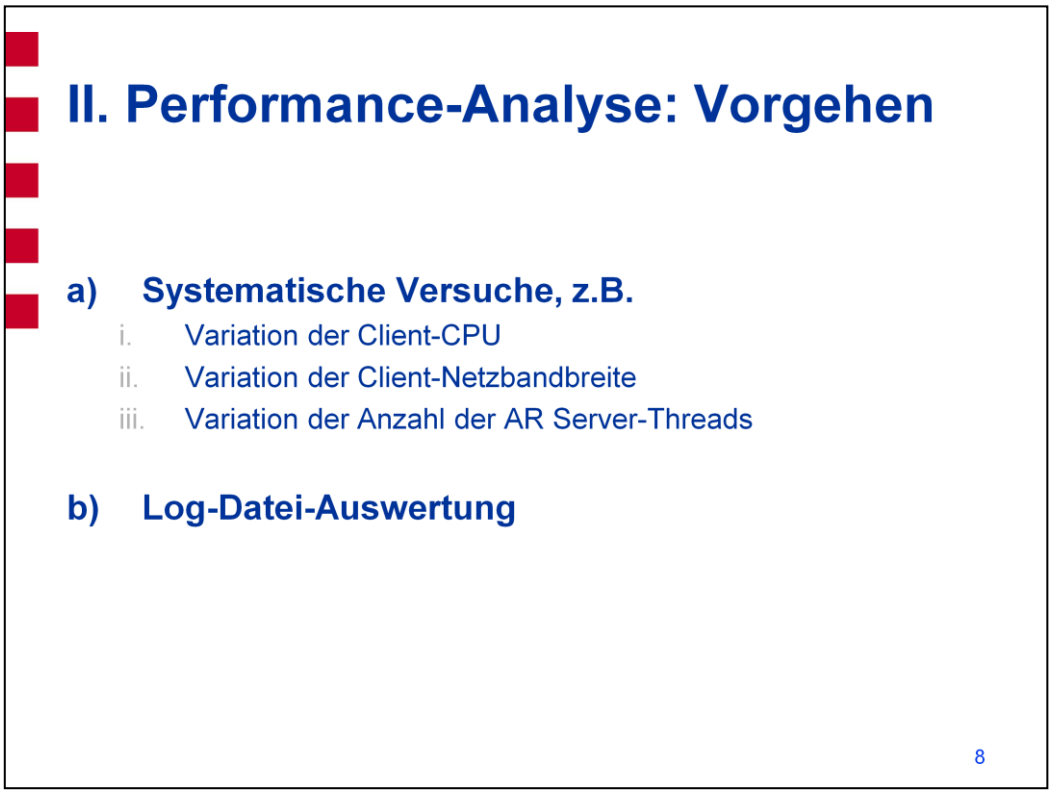
I Einführung
- Performance-Messungen
- Ursachen schlechter Performance

II Analyse
- **systematische Versuche,**
- **Log-Datei-Analyse**

III Tuning
- Datenbank
- Server
- Workflow

IV Zusammenfassung

V Diskussion



Teil II des Vortrags: die **Analyse**

a) Systematisches Experimentieren

→ zeigen, dass dies ein **nützliches und wirkungsvolles Mittel** ist.

b) Log-Dateien (mit Zeitstempeln für jede Unteraktion)

- Tools dazu

a) Systematische Versuche

- Variation der Client-CPU
- Aktions-Auswahl aus Referenz-Scenario:

	CPU mit 0,9 GHz	CPU mit 2,8 GHz	
Aktion	Dauer (s)	Dauer (s)	Ver-hältnis
Wechsel Bearbeitungsmodus	8,4	3,4	41%
Erweiterte Suche abbrechen	2,2	0,8	37%
Speichern im Bearbeitungsmodus	8,7	3,0	35%
Fokus setzen im Bearbeitungsmodus	5,5	1,8	32%
Umzug abbrechen	3,8	1,2	32%
Umzugsdialog schließen	3,4	1,0	28%
Aktivierung aufrufen	6,4	2,7	42%
Aktivierungsdialog schließen	3,5	0,9	25%
Austausch abbrechen	3,7	1,4	38%
Austausch schließen	3,7	1,1	29%
Ersterfassung abbrechen	3,0	0,8	28%

Ausgewählt

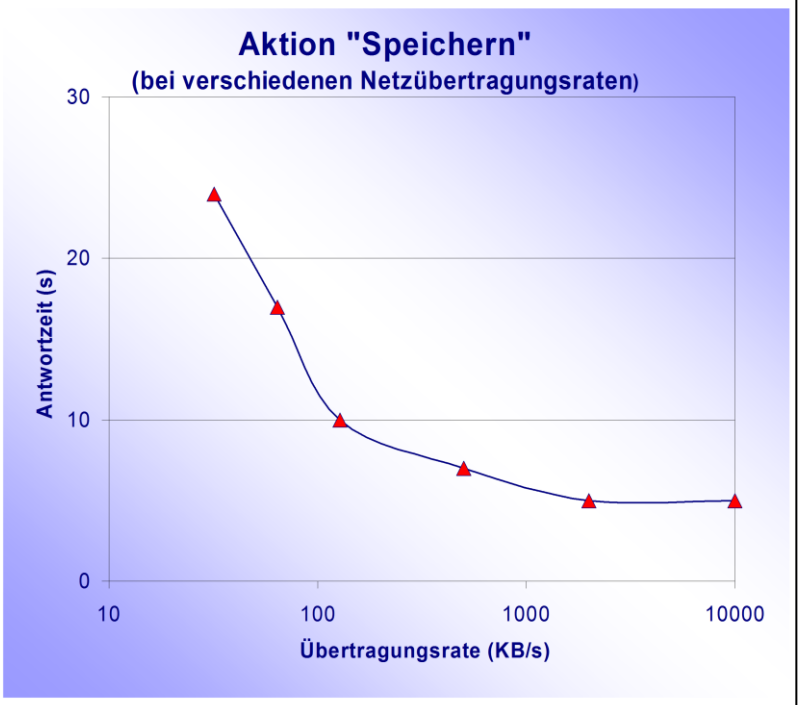
- Client-lastige Aktionen mit > 2s Dauer (900 MHz)

Anmerkung:

Rechnerisch: 900 / 2800 = 32%

Beim 2,8 GHz-PC sind in der Architektur noch mehr Komponenten schneller (nicht nur die CPU)…

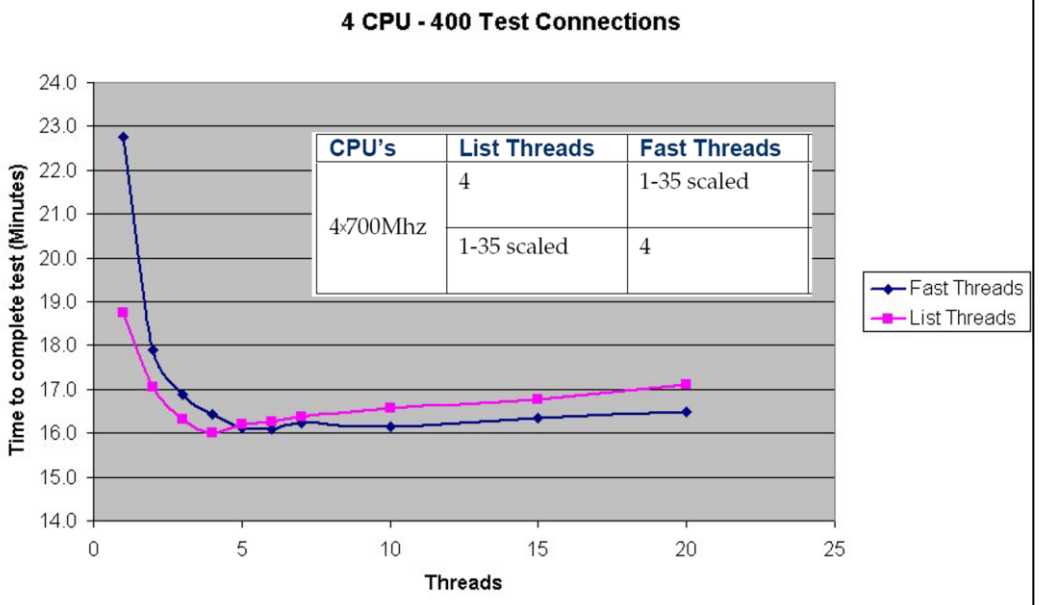
- Variation der Client-Netz-bandbreite



10

Ab 1024 kB/s in diesem Beispiel keine weitere Verbesserung…

Variation der Zahl der Server-Threads



CPU's	List Threads	Fast Threads
4x700Mhz	4	1-35 scaled
	1-35 scaled	4

Aus „Remedy White Paper: Optimizing AR System Performance with Server Thread Settings"

Entnommen aus…

<Server-Thread erklären> „Warteschlange für schnelle Anfragen/aufwändige Anforderungen"

Was wurde gemessen?

•Ein Scenario (ähnlich unserem Referenz-Scenario) wurde entworfen

•Das Scenario wurde automatisiert („Load generation tool" / „Software Testing Package"), „internal tools"

•Dasselbe Scenario wurde für 400 Benutzer gleichzeitig durchgespielt.

•Die Gesamtdauer wurde gemessen → 20 Minuten

Zunächst: 4 List Threads fix – Messpunkte für verschiedene Anzahlen von Fast-Threads (blau)

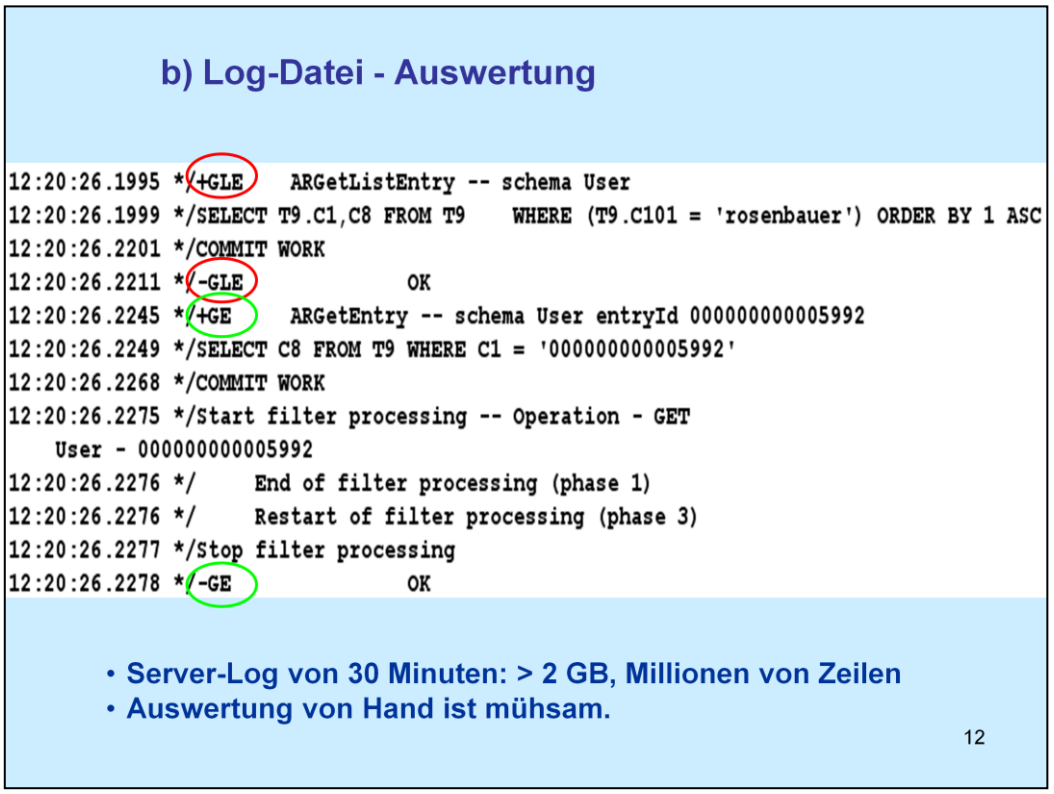
Dann: 4 Fast Threads fix – Messpunkte für verschiedene Anzahlen von List-Threads (violett).

→ ca. 7 list und 5 fast threads optimal bei diesem System

→ Eher „etwas zu viel" als „etwas zu wenig"

→ **Derartige Experimente sind ein geeignetes Mittel**, um ein Optimum zu finden (manchmal das einzige Mittel).

Sie werden auch von Remedy selbst angewandt.



Log-Datei des Servers
(ähnlich auch Mid-Tier Log, jedoch ohne SQL und Filter)

Zu sehen:
•Bereich < 0,1 s Dauer, 1 einziger Thread
•2 API-Aufrufe: GLE und GE (nach Ausführen einer Suche im UT)
•Ausführungszeiten haben 0,1 ms Genauigkeit

Wir können analysieren:

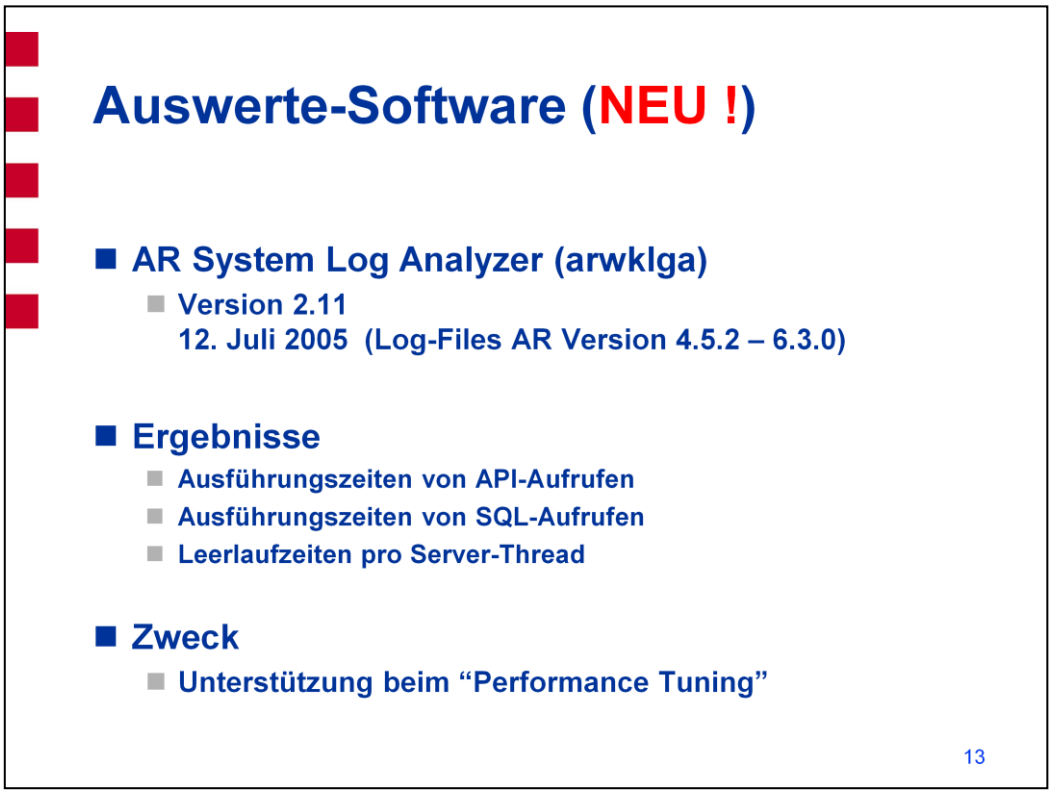
GLE
API-Aufruf: 21,6 ms, davon SQL: 20,2 ms (→ AR Server: 1,4 ms)

GE:
API-Aufruf: 3,3 ms, davon SQL: 1,9 ms (→ AR Server: 1,4 ms)

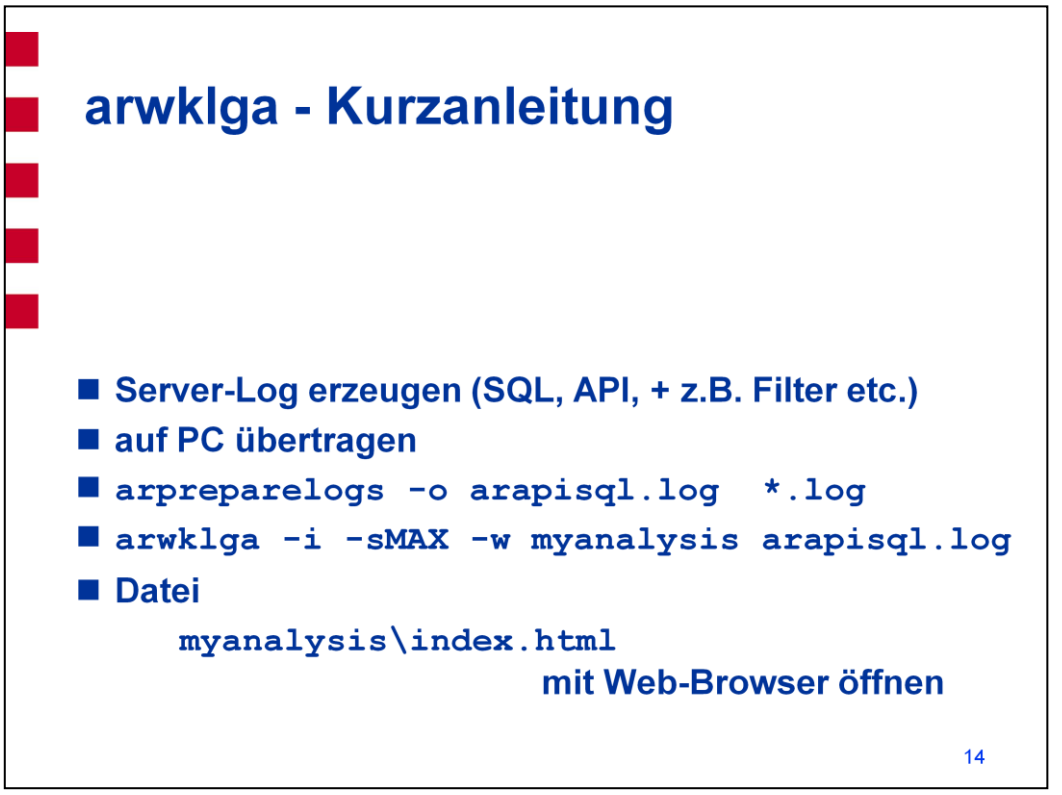
Manuelle Auswertung mühsam.

Bisher:
Erstellen von PERL-Skripten zur Log-Datei-Auswertung (ich habe eine ganze Reihe davon)

Seit 4 Monaten ist das jetzt viel einfacher und komfortabler geworden: ….

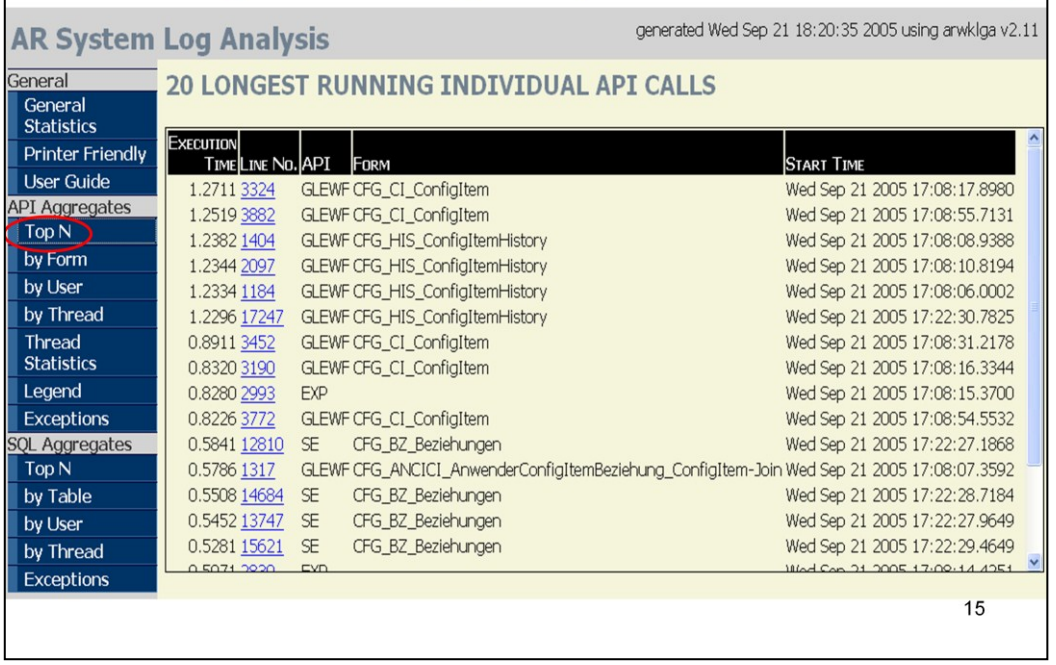


Für die Interessierten hier noch kurz wie‘s geht….



Filter/Eskalationen mitloggen – sonst Identifizierung der Stellen im Workflow später schwierig.

arwklga – Output (HTML-Version)

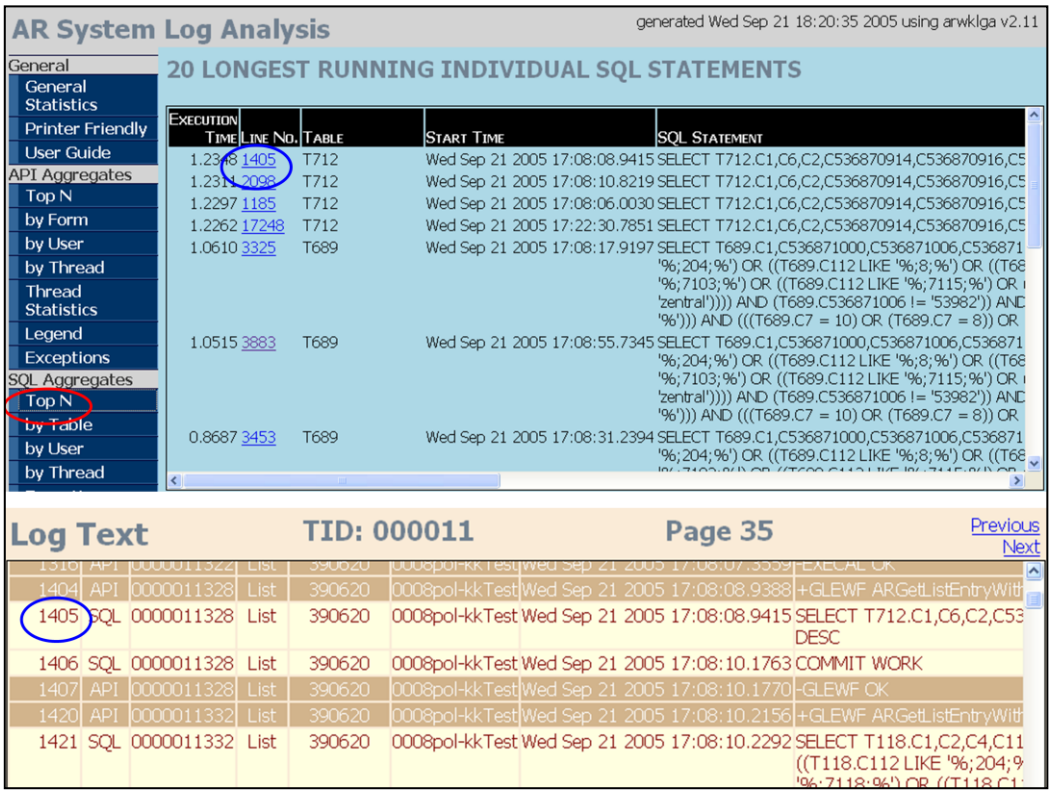


Sehr übersichtliche Darstellung,

Hier gewählt: „API Aggregates, Top N“

…Genauso geht z.B:

…„SQL-Aggregates, Top N“…



(Klick auf „Line No.“):
Durchklicken auf die Log-Datei…

Mit diesen Informationen (insbesondere mit dem Zeitstempel)
•z.B. in die Original-Log-Datei sehen
•und den zugehörigen Filter identifizieren (mittelschwer)
•Oder den zugehörigen Active Link suchen (manchmal sehr schwer).

Maßnahmen
z.B. Beseitigung/Änderung der ungünstigen Abfrage,
Index setzen auf Abfrage-Feld
(Rekonstruktion über Datenbanktabellen „arschema“ und „field“)
Oder: Tuning der Datenbank

Übersicht

I Einführung
- Performance-Messungen
- Ursachen schlechter Performance

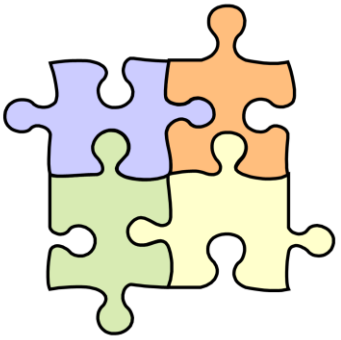
II Analyse
- systematische Versuche,
- Log-Datei-Analyse

III Tuning
- ***Datenbank***
- ***Server***
- ***Workflow***

IV Zusammenfassung

V Diskussion

III. Performance-Tuning



- **Tuning ist Prozess, nicht *„einmalige Maßnahme“***
- **Teamarbeit:**
  - Datenbank-Tuning-Experten
  - Unix-Experten (mit Analyse-Tools)
  - Evtl. Netzwerk-Experten
  - AR System Experten
  - Anwender / Kunden

18

Je nach Ergebnis der Analyse:

Start z.B. bei der Datenbank

Oracle Tuning-Maßnahmen

- Setzen von Indices
- Cachen einzelner Tabellen
- Datenbank-Reorganisation
- Shared Pool – Verkleinerung
- Tabellen-Partitionierung

Setzen von Indices…

19

häufig die einfachste und wirkungsvollste Maßnahme: **Index setzen**

CACHE-Attribut

- SGA = Gesamter Reservierter Hauptspeicher von Oracle (System Global Area)
- Buffer Cache, mit Ordnung:
  - Anfang: MRU („most recently used“) und
  - Ende: LRU („least recently used“)

Anwendung:
***CACHE* - Attribut**

Bei wichtigen, nicht zu großen Tabellen setzen!
… zwingt wichtige Tabellen permanent in den Cache

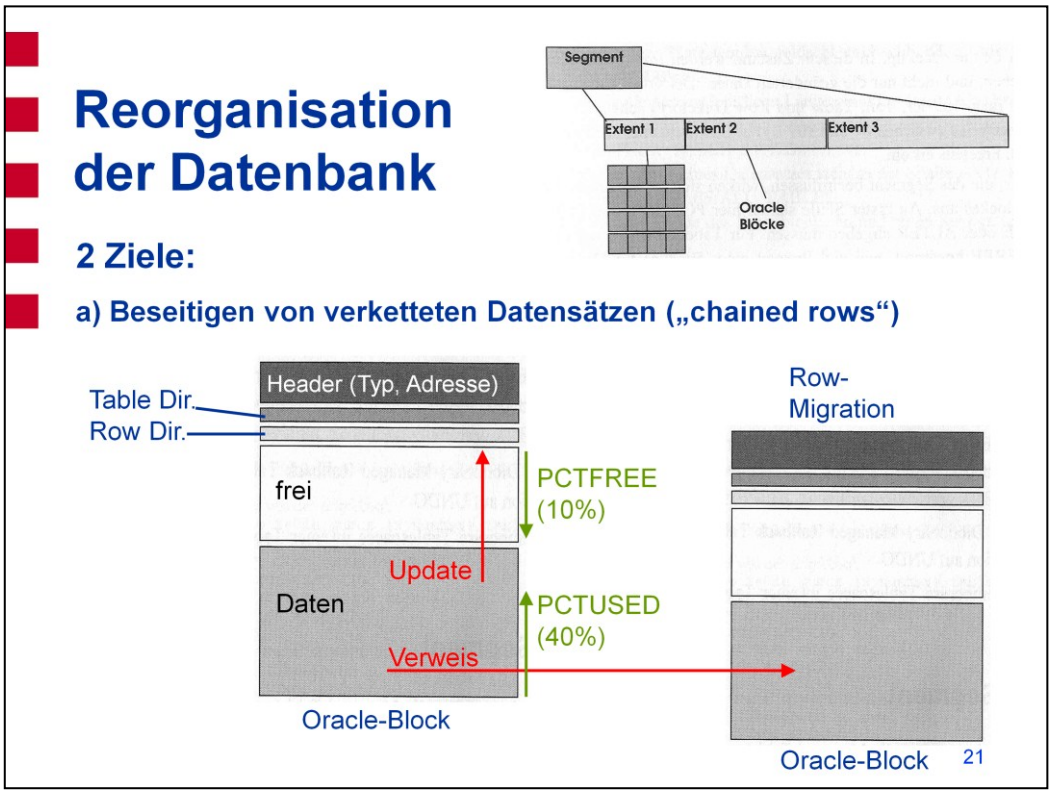
→ evtl. großer Performance-Gewinn

Gelesene **Oracle-Blöcke** werden im **Buffer Cache** gecached.

Über Index eingelesene Blöcke kommen an das MRU-Ende,
über „Full Table Scans“ gelesene kommen gleich an das LRU-Ende

Nicht wiederbenutzte Blöcke wandern vom MRU- in Richtung des LRU-Endes und fallen schließlich wieder aus dem Cache heraus.

Manuelle Beeinflussung der Oracle-CACHE-Verwaltung: CACHE-Attribut.



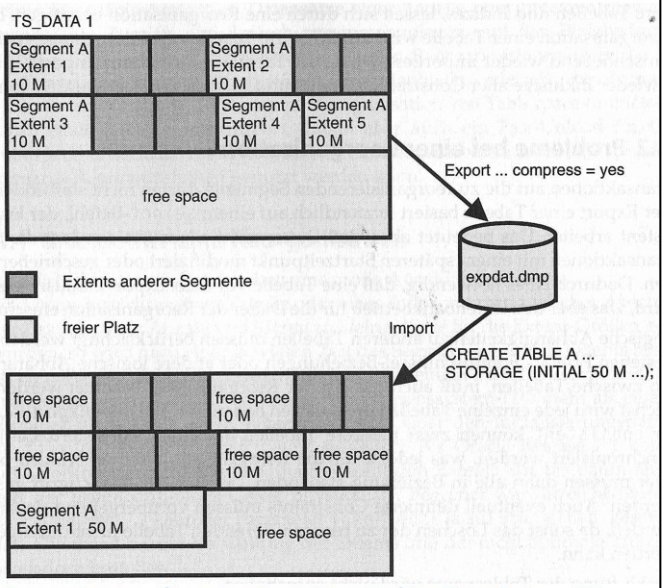
<Block erklären> (Segment=Tabelle, Extents, bestehen aus Blöcken)

Auffüllen eines Blocks mit Inserts (Submit)

PCTFREE-Parameter pro Tabelle setzbar

ardb.conf

b) Zusammenlegen von Extents



Anmerkung: „Segment“ steht z.B. für „Tabelle“. 22

Zur Reorganisation:

Export (Dump) notwendig.

Anschließender Reimport mit neuen „Insert“-Vorgängen (Auflösung verketteter Blöcke).

Ähnlich wie Disk-Defragmentieren in Windows

Partitionierung von Tabellen

- Partition-Key
- Verteilung von Tabellenzeilen auf Partitionen
- AR System „sieht“ normale Tabelle
  - weiß nichts von deren Partitionierung
- Abfragen
  - lesen nur spezifizierte Partition
  - „partition elimination“ / “partition pruning“
- Skalierbarkeit der Anwendung:
  - Full Table Scan → Full Partition Scan

Große Tabellen partitionieren z.B. nach
• Status (Incident Management) oder
• Mandant/Kunde (Bestandsführung)

Vorsicht bei Join-Bedingungen mit partition keys!

Bei Partitionierung nach „Kunden“ bleibt die Anwendung skalierbar

→ Neue Kunden mit „nochmal 150000 Datensätzen“ verlängern die Suchzeiten für Altkunden nicht.

Vorsicht bei Join-Forms!

Join-Abfragen über mehrere indizierte Felder gehen im Optimalfall zunächst nach dem Index, der die Treffermenge am stärksten einschränkt.

Bei Verwendung eines Partition-Keys in Abfrage und Join-Bedingung entscheided der Optimizer evtl. zugunsten des Partition-Keys als erstem Index, obwohl dieser die Datensätze evtl. nur auf 150000 „einschränkt“, und joined danach mit der zweiten Tabelle

→ Aus 0,3 s Suchzeit wurden bei uns 12 Minuten Suchzeit !!!

SQL-Ausführung

- Phasen
  - Query-Analyse (Parsing)
  - Berechnung des Ausführungsplans
  - Ausführung
- Ausführungsplan
  - Erstellung durch „Optimizer“
  - Speicherung (mit SQL) im **Shared Pool**
  - Wiedererkennung nur bei exakt identischem SQL
  - Sonst Voraussetzung zur Wiedererkennung: Bind-Variablen.

- SQL des ARS benutzt keine Bind-Variablen
- **Shared Pool** wird für ARS-Anwendung fast immer ohne Ergebnis durchsucht ☹.

→ *Shared Pool **klein** machen, um nutzlose Suchen abzukürzen*

24

Der **Shared Pool** cached SQL-Anweisungen mit Ausführungsplan des Optimizers.

„rule based“ (RBO)

Costbased (CBO)

Erfolg der Datenbank-Maßnahmen (*)

a) Datenbankverkleinerung von 7,8 GB auf 5,9 GB

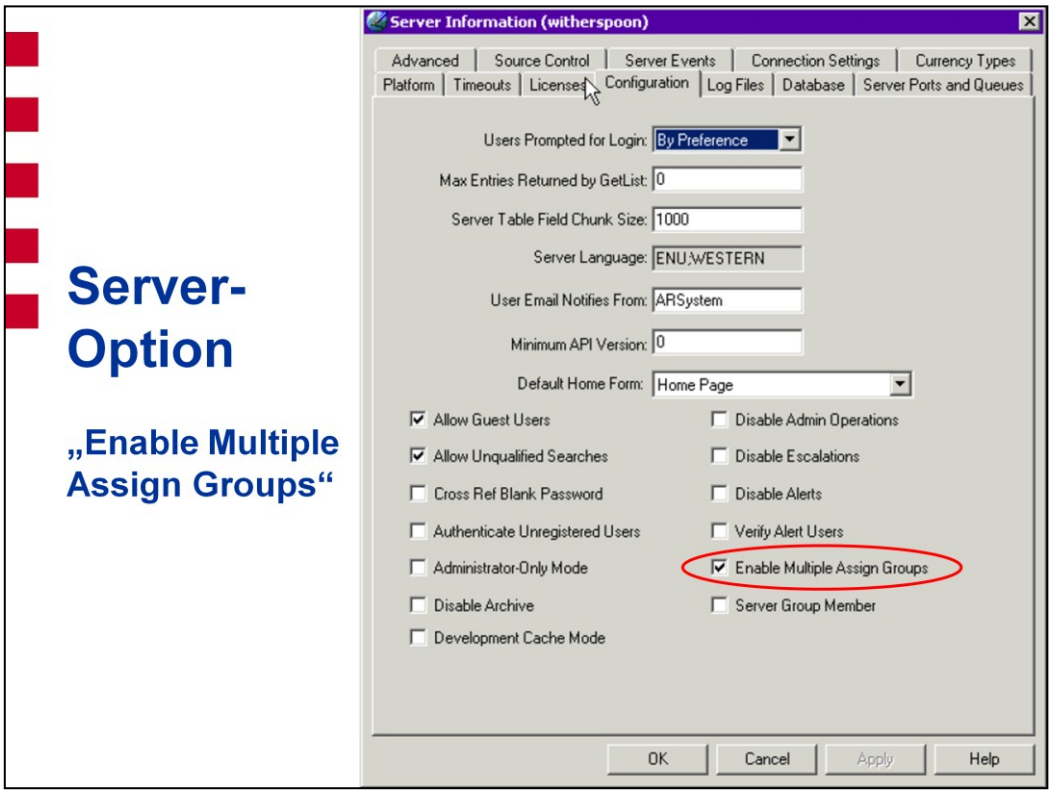
b)

Aktion	Vor Reorg (s)	Nach Reorg (s)	Verhältnis
Anmelden	**18,8**	**18,7**	99%
Erweiterte Suche aufrufen	**4,4**	**2,5**	56%
Umzugsdialog aufrufen	**8,2**	**6,0**	74%
Aktivierung aufrufen	**5,6**	**4,3**	77%
Austauschdialog aufrufen	**9,7**	**5,3**	54%
Austausch durchführen	**2,6**	**2,0**	77%
Ersterfassung aufrufen	**5,7**	**4,0**	69%
Gesamt	54,9	42,7	**78%**

*) Gewertet wurden alle Referenz-Scenario-Aktionen, die
- länger als 1,7 s dauerten und
- *keine* starke Client-CPU-Abhängigkeit haben
(Verhältnis „t (2,8 GHz) / t (0,9 GHz)“ > 66%)

AR Server und Workflow Tuning

- Multiple Assign Groups
- Localize Server - Option
- Workflow-Anpassungen



Bei Zugriffsrecht-Zuteilung auf Datensatz-Ebene („jeder Kunde sieht nur „seine“ Datensätze“):

1 Datensatz kann bequem auch für mehrere Kunden (bzw. Gruppen oder Rollen) sichtbar gemacht werden

```
SELECT C2,C3,C4,C5,C6,C7,C8,0,C112,C800000001,C800000002,C1
FROM T47
WHERE C1 = '000000000000001'
AND
(T47.C112 IN (';8001;',';8000;',';-8001;',';0;',
                                ';''Benutzer1'';')))
```

…wird mit Option „Enable Multiple Assign Groups“ zu

```
SELECT C2,C3,C4,C5,C6,C7,C8,0,C112,C800000001,C800000002,C1
FROM T47
WHERE C1 = '000000000000001'
AND
((T47.C112 LIKE '%;''Benutzer1'';%') OR ((T47.C112 LIKE '%;0;%')
  OR ((T47.C112 LIKE '%;-8001;%') OR ((T47.C112 LIKE '%;8000;%')
  OR (T47.C112 LIKE '%;8001;%'))))
)
```

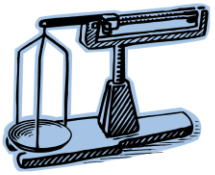
→ Benutzung eines Index damit in vielen Fällen unmöglich !

- **Ebenso bei „Dynamic Groups“ (60000-60999)**

28

Immer prüfen, ob diese Option verzichtbar ist, z.B.

- **bei Altanwendungen (geschrieben unter AR System Versionen 4 oder 3)**
- **bei neuen Anwendungen, durch Gebrauch mehrerer „Dynamic Groups“ mit je 1 Eintrag anstelle von 1 „Assignee Group“ mit vielen Einträgen**

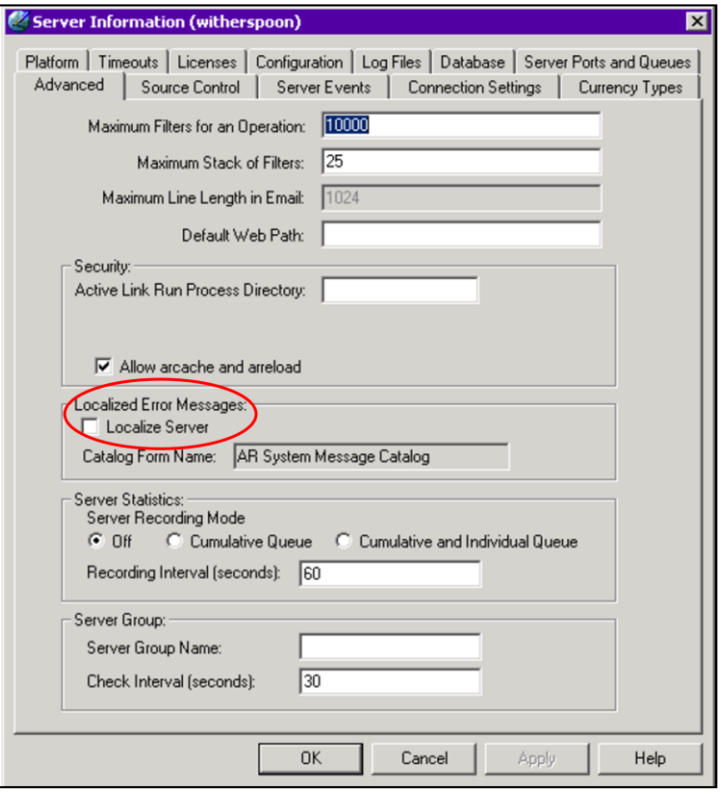


29

Ansonsten bleibt nur die Tabellen-Partitionierung.

Server-Option

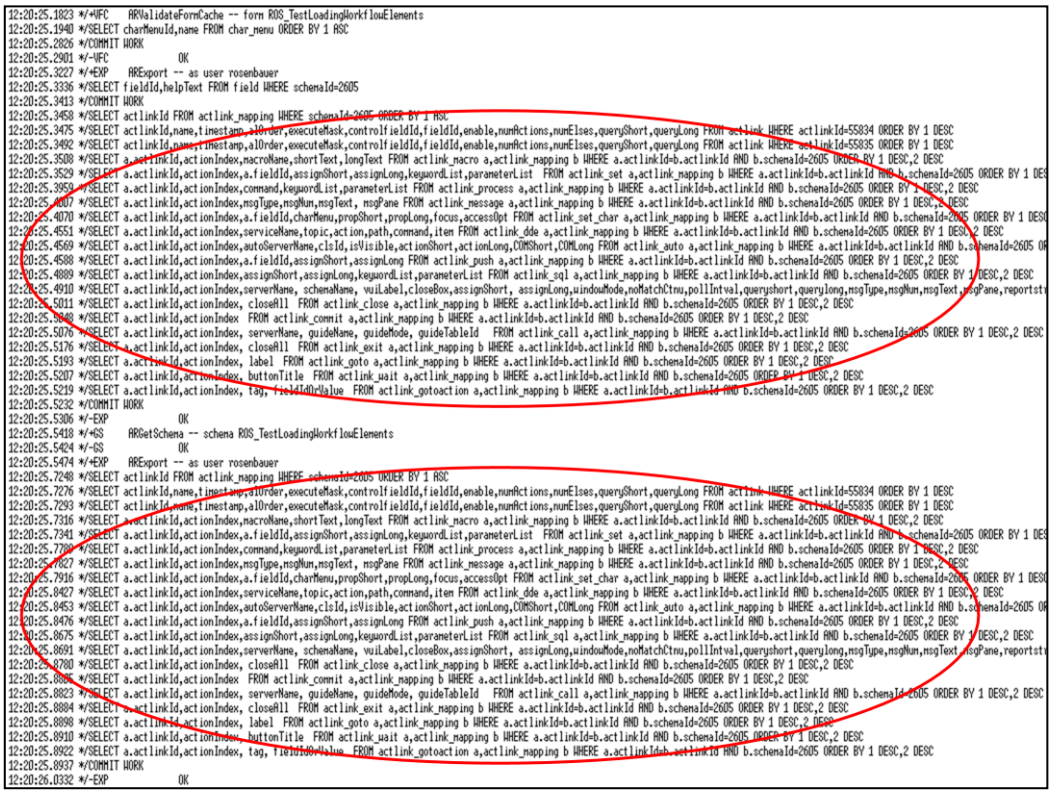
„Localize Server“



Ermöglicht es,

•AR System Fehlermeldungen („no server available“) zu verändern/ zu übersetzen

•Eigene Meldungen („Messages“) in mehreren Sprachen zu hinterlegen



Demo-Beispiel: Einfache Form mit 2 Active Links mit jeweils 3 Aktionen

- **Mit „Localize Server“: Alle Abfragen zweimal identisch**
- **Pro Active Link und pro Menü: ca. 1 zusätzliche SQL-Abfrage**
- **Caching im User Tool ebenfalls zweifach:**
  - Einmal allgemein
  - Einmal lokalisiert
- **Antwort von Remedy (Ticket-ID ISS01052967, noch offen):**

> Localized messages are downloaded to the local client as though they were another view (localized) of the form. This view definition then contains the active link, and its particular text message.
> (…) performance impact, however (…) would be expected to be slight/unnoticable

- **Localize Server – Option: evtl. verzichtbar?**

32

Nach Änderungen (update cache):

Bei 1000 Active Link- und Menüobjekte / Anwendung - ca. 1000 unnütze SQL-Abfragen pro Benutzer beim Öffnen einer Form

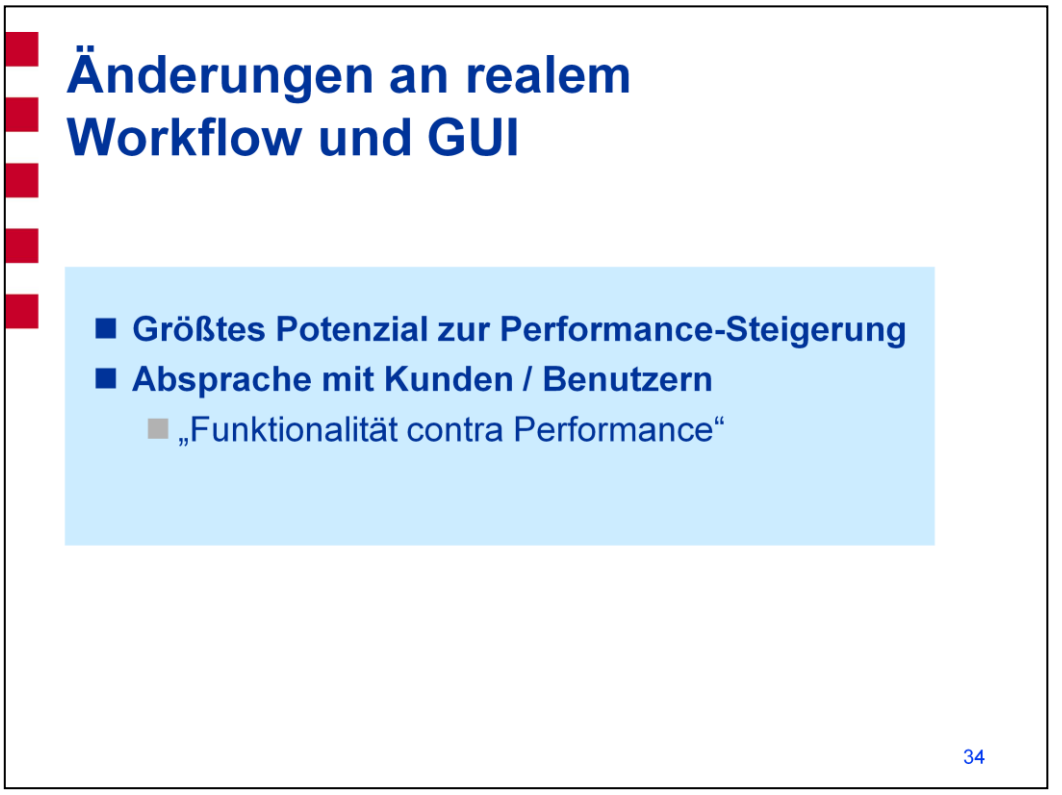
bei 300 Benutzern damit am nächsten Morgen: 300.000 überflüssige Abfragen zwischen 7.30 Uhr und 9:00 Uhr

→mehr als 50 zusätzliche Abfragen/s an Datenbank.

(alle 18 Sekunden ein neuer Benutzer)

Workflow-Programmierung

- **Verbesserungspotenzial**
  - Zu große / zu viele Hintergrundbilder / Icons
  - Häufige Aktualisierung von Tabellen, sogar von unsichtbaren
  - Fehlende Indices / kombinierte Indices
  - Set-Fields-Aktionen in Filtern über „$PROCESS$ - Aufrufe („Blocking Actions“)
  - Überflüssige Temp-Felder, Deaktivierte Active Links
  - Vergessene „`$“ im Namen von Active Links mit Table Loop Guide - Aufrufen
  - Push-Field-Aktionen mit Bedingung „1 = 0“ entfernen
  - ($TIMESTAMP$ - ‘Create Date‘) > 3600 umschreiben
  - Feldlängen < 4000 Zeichen verwenden
  - Erlauben von „unqualifizierten Suchen“ (Server-Option)



z.B. Tabellen-Refresh nur nach Klick auf Tabelle, nicht bei Öffnen des Fensters

Übersicht

I Einführung
- Performance-Messungen
- Ursachen schlechter Performance

II Analyse
- systematische Versuche,
- Log-Datei-Analyse

III Tuning
- Datenbank
- Server
- Workflow

IV Zusammenfassung

V Diskussion

VI Zusammenfassung

- **Ursachen für schlechte Performance**
  - Ungünstige Workflow-Anforderungen
  - Client-CPU, Mid-Tier-Server, Netz, Firewall
  - AR System Server, Server-Netz, Datenbank
- **Tuning**
  - Ursachenanalyse
  - Maßnahmen, z.B. Datenbank
    (Indices, CACHE, Shared Pool, Reorg, Partitioning)
- **Performance / Skalierbarkeit**
  - Von Anfang an im Workflowdesign / im Datenmodell berücksichtigen
- *Performance nicht um jeden Preis !*

36

Abwägen zwischen Funktionalität und Performance !

Abstürzende Server-Prozesse

Blocking Actions – genaueres

Mid-Tier – Ausführung von Active Links

Flashboards

Top und RAM-Angabe

Identifizierung von Anwendungsschritten, die besonders lange dauern, Analyse der zugehörigen Workflow-Vorgänge und gezielte Optimierung des Workflow-Codes für diese Aktionen

Identifizierung von ungünstigen Datenbank-Abfragen, sowohl mittels Analyse von Log-Dateien im ARS und von SQL-Logs in der Datenbank als auch durch Auswertung von Statistik-Informationen innerhalb der Oracle-Datenbank

Aufteilung in Table Spaces nach Anwendungen – schafft Übersicht – siehe auch ardb.conf

Regelmäßige Reorg

„Related Workflow“ unter 5.1.2 : Datenbanktabellen von Zeit zu Zeit „aufräumen“

SELECT T539.C1,T539.C1,T539.C1 FROM T539 WHERE (1 = 2) ORDER BY 3 ASC

Perl-Script zum Auffinden der Active Links