Knowledge Management in Autonomic Database Performance Tuning

David Wiese
Friedrich-Schiller-University Jena
Database and Information Systems Group
07743 Jena, Germany
david.wiese@uni-jena.de

Gennadi Rabinovitch
Friedrich-Schiller-University Jena
Database and Information Systems Group
07743 Jena, Germany
gennadi.rabinovitch@uni-jena.de

Abstract — Databases are growing rapidly in scale and complexity. High performance, availability and further policy goals need to be satisfied under any circumstances to please customers. In order to tune the DBMS within their complex environments highly skilled database administrators are required. Unfortunately, they are becoming rarer and more and more expensive. Hence, improving performance analysis and moving towards automation of problem resolution requires a more intuitive and flexible source of decision making. This paper points out the importance of knowledge for autonomic database tuning, proposes a component-based knowledge model and briefly presents first prototypical evaluation results.

autonomic database tuning, self-management, MAPE, performance, problem resolution, tuning knowledge, resource dependencies, optimization

I. INTRODUCTION AND MOTIVATION

Modern database-oriented OnDemand environments are coined by a heterogeneous diversity of components, architectures and applications. In order to increase productivity and efficiency, decrease the total cost of ownership (TCO) and to meet business requirements, Performance Management, in particular database tuning, has become of fundamental importance. It involves the collection, storage and management of measurement data in order to enable resource monitoring, tuning and optimization, early problem diagnosis and repair. Achieved DBMS’ performance depends on database administrators’ (DBAs) individual skills, home-grown tuning scripts and is in most cases reactive to obvious and urgent performance problems [11]. DBAs face increasingly more challenges due to the growing complexity and must be knowledgeable in areas such as capacity planning, physical database design, systems tuning and systems management. Accordingly, the operation of a DBMS environment is a costly matter with remarkable effect on the TCO. This cost factor even increases with the steadily growing demand for DBMS and the number of attached applications.

The basic idea, therefore, is to let the system deal with the complexity of performance management and disburden DBAs to achieve minimal to no user intervention. IBM terms the field and the computer science wide push towards self-managing systems as Autonomic Computing [7]. To be self-managing (especially self-optimizing), an autonomic system has to comply with an automated, potentially infinite, iterative feedback-based control loop paradigm [7]; to monitor resources and collect the details it needs; to analyze those data in order to determine if something needs to be changed; to create a plan (as sequence of actions) that specifies the necessary changes; and to execute those actions.

This paper provides a basis for classifying, formalizing, obtaining, storing, maintaining, exchanging and individually adapting DBA expert tuning-knowledge as shared domain of understanding in the autonomic management process of a relational database management system in its environment. For this purpose, we propose a component-based knowledge model which is vital for the construction of an autonomic management engine capable of realizing the process of autonomic database performance tuning such as finding or predicting performance problems, determining their root causes and finally resolving these problems.

Section 2 points out the importance of knowledge, its acquisition, storage and utilization within the context of autonomic database performance tuning. In Section 3 we propose a component-based knowledge architecture required for the construction of an autonomic management engine capable of realizing the MAPE control loop and give a deep insight in each of the disjoint knowledge categories. In Section 4 we briefly summarize and evaluate how those four MAPE phases can be implemented to enable workload-oriented and best-practices-based autonomic tuning. Finally, in Section 5 we give brief conclusions, present our current work and outline directions for future work.

II. KNOWLEDGE - THE KEY TO MAPE

According to [8] knowledge in self-managing systems is more than simply a collection of data and facts. It differs from data or information in that new knowledge may be created from existing knowledge by learning and logical inference. Moreover, it is used to carry out processes, especially processes that can be automated. The task of automating how to capture and select key characteristics of managed resources, detect (relevant) changes, identify problems, isolate their root causes, finally schedule or initiate actions to correct any problematic behavior or even better predict and avoid complex problems, as well as learning from past experiences, requires in-depth expert and process knowledge and is central to MAPE.

The logical and architectural foundation of all MAPE phases is a common knowledge element that is shared among the four MAPE parts and represents a formalized understanding of the problem space. Aspects like topology information, policies and historical data are relevant to the problems that ought to be detected, diagnosed and finally resolved with an autonomic management engine. Such collaborative problem domain information facilitates integration, interoperability and machine processability and, thus, contributes to disburdening DBAs. Therefore, constructing an autonomic infrastructure also demands tools, techniques and processes to efficiently formalize, store and manage (an organization’s) intellectual assets and current best-practices.

Plenty of unstructured and non-formalized information about database tuning best-practices is available online and information resources range from traditional news groups to IT blogs, online magazines, IBM Redbooks publications, and product manuals. However, no guidelines exist about what kinds of knowledge to collect, how to represent data about problems in the domain and how to resolve a problem without the need for a high level of human intervention.
In this article we, therefore, propose a generic model with blocks that are needed to construct a profound knowledge base on a rather abstract conceptual level and then give some detailed insight into topological inter-resource dependencies. The main focus of applying such a knowledge model lies on the assistance of the autonomic analysis and planning phase.

III. A KNOWLEDGE MODEL FOR AUTONOMIC DATABASE TUNING

Following and extending the ideas of IBM’s Architectural Blueprint for Autonomic Computing [7], the knowledge of an autonomic tuning architecture as a whole should provide a shared and common understanding of the environment and the problem space. We propose to divide our conceptualized architecture into an object level (environment under control) and a meta level, representing the types and containing components of knowledge. Figure 1 depicts the layered structure of the corresponding knowledge components built on top of each other, in which more general knowledge is at the bottom level and more specific at the top. Each building block utilizes and enriches the knowledge it is founded on, masking or emphasizing details for a specific purpose.

One of the most important requirements is the need to keep track of all changes (changes due to workload or due to autonomic tuning actions) to the data. Therefore, a set of change history repositories exists in the according knowledge layers. Changes to the elementary system objects can be protocollled indirectly by storing a sensor history or directly by logging tuning actions and changes in workload characteristics for example.

We will now briefly describe the layers of knowledge and their subcomponents. Hereby, we will concentrate on the Topology Knowledge.

A. Environment Under Control (EUC)

The foundation of our proposed knowledge model is the "Environment Under Control" object level. It represents the workload and the (database management) system which is composed of (atomic) resources. All other layers are built on top of it and respectively try to point out certain knowledge specifics of the controlled environment.

B. Policy Knowledge

Policies govern the decision making process for autonomic managers and indicate which resources to monitor as well as whether or not changes need to be made to the system. In a fully autonomic system goal-based policies are the only elements provided by the administrators expressing resource-based and system-wide (aggregated) high-level or low-level goals (thresholds, operational states) and performance expectations. The model contains sensors and effectors with corresponding policy data. It describes desired fuzzy values/value ranges of sensors and acceptable cost of each value/value range, as well as acceptable value ranges of some effectors (some effectors might be fixed, for some others a possible change up to +/-10% might be acceptable, etc.) depending on the workload.

C. Workload Knowledge

Database performance tuning is strongly related to the understanding of the database applications as well as their resulting workload effects on the system’s behavior. Hence, the workload also plays a major role in Figure 1. Without it, no system would need continuous and adaptive tuning. It can be considered as the surrounding and influencing environment of the system which has to be managed autonomously.

We consider the workload as non-modifiable and exposed to permanent changes (variation of types and frequencies of transactions). Workload characteristics, however, can be determined indirectly by a set of measurements of the DBMS sensor interface [1, 12].

Workload does not only affect the system’s behavior tremendously, it also biases knowledge model instantiation by affecting the topology dependency model, problem diagnosis and resolution. Workload Knowledge is based upon the workload object level and contains workload classes, characterization rules, thresholds and possible effects on the observed environment and on the other knowledge layers. Workload can be classified in transaction classes with different associated performance goals, resource requirements, system object dependencies, tuning plans, events and sensor data to monitor. Since all metadata has to be stored for each of the workload classes it is advisable to keep the number of workload types low and distinctive.
D. Problem Diagnosis Knowledge

The Problem Diagnosis Knowledge addresses the best-practices usually used by a system administrator to detect problems, interpret the status and identify the root cause(s) of a problem resource using (current and past) sensor values and information on dependencies.

Every static (critical set of value combinations) or dynamic (change in system state) situation that may require an automatic reaction of the system can form a primitive/atomic event that occurs at a specific point in time. Examples of atomic events that are based upon sensors are exceeded thresholds or specific performance metric values. In addition, user complains can be treated as events as well.

Complex events are part of the Event Correlation Knowledge and can be constructed recursively (through operators like conjunction, disjunction, sequence, negation, reduction, etc.) from atomic or complex events with the help of an event algebra.

Our event tuning plan subscription concept identifies all TPs associated with a specific event (many-to-many relationship).

In order to support workload-dependent problem diagnosis we define a set of thresholds per workload type. Figure 2 shows an exemplary definition of an atomic and a corresponding complex event for the key performance indicator "rows read/rows selected". If the appropriate threshold is exceeded the corresponding tuning actions are executed. All subsequent threshold exceedances of the same type are then ignored for 5 minutes in order to avoid hasty overreactions to frequent problem notifications.

Figure 2. An Event Definition

Further examples of how event algebra can be used in the area of autonomic database tuning are presented in [12].

E. Problem Resolution Knowledge

Events are used to trigger one or more tuning plans (TP). The latter are an encapsulated sequence of tuning actions, enriched with parameters and metadata. So, informally, a TP can be considered as a reaction to past, current or upcoming problems (indicated via events or by prediction) in pursuance of one or many goals (policies). TPs are able to read out sensor data and access dependency knowledge in order to implement decision logic.

Figure 3 depicts one of our tuning plans that automatically applies index recommendations of the DB2 Design Advisor for the worst five select statements. Based on the ratio "rows read/rows selected" [4], i.e. how many rows had to be read to find the target rows, the tuning plan determines the top-5 SQL statements from the DB2 statement cache. Thereby, it ignores all non-select statements as well as all "SYS%" tables and views, since they cannot be indexed. The determined top-5 statements are saved in a file that is provided as input for the DB2 Design Advisor [4]. Subsequently, the indices suggested by the Design Advisor get automatically created by executing the DB2 DDL batch script. Depending on the workload type, the Design Advisor is called with different parameters. For OLTP workloads, advise time is set to its minimum and only indices will be recommended. For OLAP workloads, it seems useful for the advisor to have more time for index calculations and to recommend materialized views as well. The execution of this tuning plan is expected to drastically reduce the ratio "rows read/rows selected" by using newly created indices efficiently and thereby increasing performance.

Table 1 - DB2 SQL Mon/sql

<table>
<thead>
<tr>
<th>Event Definition</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLTP threshold: rows read/rows selected &gt; 5</td>
<td>rows, UCASE(stmt_text) as stmt_text FROM db2pm_1.dynsql WHERE DB_NAME = &quot;&lt;db_Name&gt;&quot; AND UCASE(stmt_text) not like '%ADVIS%' AND UCASE(stmt_text) like 'SELECT%' AND UCASE(stmt_text) not like '%TRANSACTION%' AND UCASE(stmt_text) not like '%DBMSP%' AND UCASE(stmt_text) not like 'ADVIS%' AND internal_to &gt;= (CURRENT_TIMESTAMP - 5 MINUTES) AND AVG_NUMBER_ROWS_SELECTED_ROW IS NOT NULL GROUP BY UCASE(stmt_text) HAVING max(num_executions)/min(num_executions) &lt;= ORDER BY 1 DESC FETCH FIRST 5 ROWS ONLY*, stmt_text</td>
</tr>
<tr>
<td>OLAP threshold: rows read/rows selected &gt; 25</td>
<td>WIN.write_File(result.get_row(row).get_column(&quot;stmt_text&quot;), infile) for (int row = 0; row &lt; result.get_size(); row++)</td>
</tr>
</tbody>
</table>

Figure 3. A Tuning Plan for Index Design

In addition to TPs and event tuning plan subscriptions the Problem Resolution Knowledge also contains effector and TP cost knowledge. The effector cost knowledge contains all effectors and captures the (learned or estimated) costs of updating each of them. Costs can be classified in terms of number of operations, required I/O, follow-up costs or effects on resources. For enabling a cost-based TP selection, the total costs of the execution of some specific TP can be easily determined by summarizing corresponding effector costs. Based on total costs of TP execution, the TP with least cost (and possibly average effect on other resources) can be selected. In some specific cases (for example in case of severe problems) it might be of high importance to take actions with highest impact disregarding the costs of these actions. However, it is still unclear and subject of research how to describe costs and how to calculate them. In this context we are thinking about the application of DB2 optimizer capabilities.

F. Topology Knowledge

The Topology Knowledge embodies (static and dynamic) information about the (hierarchy of) system components, their status, construction, configuration, behavior and interrelationships. It comprises layers for data value statistics, sensors and effectors as well as (functional) inter-object dependencies. The whole topology model is the main foundation for implementing autonomic detection, diagnosis, notification and rectification plans.

1) Statistics

Statistical information about the contents of the DBMS objects (tables, views, indices, materialized views, etc.) has to be kept in order to assist in problem diagnosis and resolution. Such information can be the number of rows in a table, the average row length, the use of space by a table or index, the number of different values in a column, distribution of values, integrity constraints, etc. In addition to table or index size and data distribution information, one can also collect statistical information about the cluster ratio of indices, the number of leaf pages in indices, the number of table rows that overflow their original pages, and the number of filled and empty pages in a table. Today’s relational DBMS already provide this data in non-updatable database statistics catalog tables and views.

2) Sensor Knowledge

In general, every managed resource consists of characterizing sensor elements (resource attributes) that can be used to reflect current and past (when being kept within a long-term sensor repository) state, resource allocations and performance. We differ between two types, static and dynamic sensor elements. Static elements like the name and path of a database or physical memory size are supposed to be fixed during resource’s lifetime. In fact,
they could be changed, but are in general stable and unchanging. Dynamic elements, on the other hand, are subject to fluctuant changes and their values are only valid for a single period of time. For a database these could be sensors indicating database’s health status, number of connected agents, etc.

In the world of IBM DB2 there are many proprietary tools (such as IBM Performance Expert [5]) and DB2-built-in mechanisms (such as DB2 System Monitor [6] for snapshot and event processing; db and dbm configuration, DB2 registry and log files) available to collect, update, store and analyze sensor performance measurements. Sensor meta-data does not only specify mechanisms for retrieving current and historic sensor values, but it also allows the discovery of type of data (counter, gauge, watermark, time, etc.).

Obtaining and keeping (a discrete collection of) sensor data is essential for problem diagnosis, root cause analysis and trend recognition. However, the trade-off between collection overhead and data quality must be considered as well when implementing an adaptive monitoring strategy. Collecting sensor values too frequently can impose significant overhead on the monitored (production) system. On the other hand, missing sensor values may result in underdiagnosis of acute bottlenecks of the system.

3) Effector Knowledge

Effectors are used to change the state and/or behavior of a particular resource. Current DBMS offer a set of tuning knobs and configurable parameters that enable modifications (database tuning activities) in the actual assignment of system resources.

Effectors in DB2 can be classified by the object and type of modification as well as by the command required to take effects. There are the following effector adaptations possible in DB2: adjusting environment variables, changing configuration parameters (db/dbm), modifying configuration files (db2nodes.cfg, db2cli.ini), updating registry variables or creating/altering/deleting/restaring elements of the (physical) DBMS layer (such as indices, materialized views, instances, bufferpools, etc.). Some of the commands required for effector control are subsumed below:

- DB2 SQL statements (e.g. alter bufferpool, create index)
- DB2 Command Line Processor commands (e.g. runstats, reorg, (re)bind, update db/dbm cfg)
- DB2 system commands (e.g. db2exp1n, db2set, db2advis, db2stop)
- Operating system commands

Metadata for configuration parameters includes their data type, value range, default values, expected impact on performance (e.g. "high", "medium", "low" as seen in [4]) and performance category (e.g. I/O, Locking, Logging, Sorting, etc.). Configuration parameters can also be classified by modifiability, range and type of values. Value ranges are discrete and can be numeric or textual. Modifiability considers effectors that are online or offline configurable, their severity, effect on system’s stability or application semantics and time frame of changes (preventative vs. reactive).

4) Dependency Knowledge

Reasons for poor performance can be manifold and frequently not obvious. Furthermore, they are often difficult to track down due to interrelationships between system components. Bottlenecks can occur in any of the resources or arise through some interactions between them. Relationships between resources are complex, not well understood and very hard to explore, model and quantify manually. Consider the following scenario: A DBMS manages 3 bufferpools $b_1$, $b_2$ and $b_3$ where $b_i$ belongs to database $DB_i$ and $b_i$ belongs to $DB_2$. A policy $p$ says: "The average bufferpool hit ratio (BPHR) of all bufferpools should be at least 92%". For each bufferpool $b_i$ $(i = 1, 2, 3)$ an event $e_i$ exists that states: "If the average of the previous 10 BPHR measurement values for $b_i$ is smaller than 92%, then raise an alarm". Reactions to this generated alarm event are two tuning plans "Increase bufferpool size for $b_i$ by 10%" and "Activate the extended storage for $b_i". During decision making in the planning phase of a MAPE cycle the following questions might be relevant, assuming that event $e_i$ has been triggered:

- Which of the both tuning plans should be executed? Are there other possible plans contributing to the problem resolution as well?
- Would it have more impact on performance to increase the bufferpool size by another value instead of 10%?
- If TP$_1$ gets executed, do the database configuration parameters "ESTORE_SEG_SZ" and "NUM.gameObject SEG" have to be modified (assuming that the extended storage is activated by altering the appropriate bufferpool)? If so, which values have to be supplied to them?
- Is bufferpool $b_i$ also affected by changes to the database configuration?
- Is there an influence on performance for $b_i$, if both "estore" parameters are set?

To support the decision making and provide answers to such questions, an inter-resource dependency model is indispensable. The model describes dependency existence and workload-dependent effect among effectors and sensors. It is built on top of the preceding topology model components and contains all sensors and effectors [9, 3]. The Dependency Knowledge distinguishes between three different kinds of dependencies among resources, as well as among their sensors and effectors:

System Objects Hierarchies: Most of the tuneable database system objects are connected via natural (implicit) hierarchical relationships. An instance in DB2 consists of multiple databases, a database comprises multiple tablespaces and each tablespace is independent of all others. Systems resources are connected via natural hierarchies. A list of environmental constraints and facts that (possibly) led to its current sensor values is associated with each resource (e.g. which applications were active on which tablespaces and issued which transactions, which database tables have been read/written or what was the configuration at that time). It can be implemented using multidimensional structures (data marts) that indicate inter-dimension member relationships with or without any measurement variables [10]. However, collecting and storing such background data which is useful for problem detection and diagnosis can become costly in terms of time and space if it is done frequently with each sensor value.

1 The bufferpool hit ratio indicates the percentage of time that the database manager did not need to load a page from disk in order to service a page request [4].
2 The parameter ESTORE_SEG_SZ specifies the number of pages in each of the extended memory segments in the database [4].
3 The parameter NUM.gameObject SEG specifies the number of extended storage memory segments available for use by the database [4].
4 Extended storage provides a secondary level of storage for bufferpools. This allows a user to access memory beyond the maximum allowed for each process. [4]
**Cause-Effect Relationship:** This type of relationship describes the levels of dependency (existence and degree) among effectors and effectors or effectors and sensors that are briefly introduced in the following:

**Effector-Sensor Relationship** describes the magnitude of changes in sensors affected by effectors under a given set of prevalent (environmental and workload) constraints and allows answering questions like:

- What will be the value of my sensor $s$ if effectors $e_1$ changes to $v_1$ and $e_2$ changes to $v_2$?
- Which effectors can be changed and to which values in order to get the sensor $s$ to the value $v$?

**Effector-Effector Relationship** embodies the correlation of effectors among each other under a given set of constraints. Two possible types of effector correlations are described in following:

- The values of effector $e_1$ are dependent on values of effector $e_2$ (e.g. to set LOGSECOND to "-1" the parameter USEREXIT must be set to "yes").
- In order to achieve maximal impact of parameter modification, it is advised or required to change certain parameters in conjunction (e.g. MAXLOCKS and LOCKLIST should be changed together).

Further work in that area has to point out common and general requirements for additional types of inter-resource and intra-resource dependencies. The required knowledge can be obtained from experiences, the DB2 documentation, by questioning DB2 experts or by applying our dependency detection tool, as described in [2].

**IV. A Prototype for Autonomic Database Tuning**

The knowledge architecture proposed in Section III is described on a rather conceptional level and therefore does not prescribe a particular instrumentation technology. In [12] we describe a possible instrumentation called Autonomic Tuning Expert (ATE) developed within the scope of an IBM CAS project. ATE’s architecture is based on widely accepted and influential technologies and industry-proven products like Eclipse, Generic Log Adapter, IBM DB2 Performance Expert, Tivoli Active Correlation Technology, and Common Base Events. These products and technologies are combined in a way to build a component-based MAPE loop for automating typical tuning tasks. The ATE infrastructure is designed to be the core component of an ecosystem that enables DBAs to design, exchange, adapt, and execute best-practice tuning methods. As a proof of concept we used ATE for tuning IBM DB2. However, the concepts developed are applicable for tuning and administration of any database-driven software stack.

During our research best-practice oriented problem diagnosis and resolution knowledge has been extracted from official DB2 documentation, white papers and Redbooks publications, and captured in tuning plans along with triggering events. Thereby, implemented tuning plans cover the areas of online resource allocation and configuration (heaps, caches, locklist), physical database design as well as statement tuning.

The results of a 105 minute tuning run are provided in Figure 5. During the first 45 minutes OLAP workload has been generated on the system. It was followed by a 40-minute OLTP and an approximate 20-minute OLAP period. After the first 5 minutes of the test run, ATE tuning was started, as indicated by a solid line. As no indices have been created at startup deliberately, the ratio "rows read/rows selected" stays rather high initially. Whenever the "rows read/rows selected" threshold (compare Figure 2) is violated, indices recommended by the DB2 Design Advisor for worst SQL statements are created, gradually improving I/O performance. By the end of the first OLAP period, there is a noticeable drop caused by creation of several indices by that time, which remain useful for the next OLAP workload period as well. Due to the nature of OLTP statements, the effects of index creation get already visible within the first few minutes. During the rest of the OLTP period, the ratio remains below the problem indicating threshold.

**V. SUMMARY, CONCLUSIONS AND FURTHER WORK**

Autonomic Computing is addressing the prevailing and increasing complexity by applying technology to manage technology with minimal human intervention. It is thereby moving the burden of managing systems from people to technologies, so that professionals can concentrate on tasks with higher value to the business.

In this paper we proposed a component-based knowledge architecture which is required for the construction of an autonomic
management engine capable of realizing the MAPE feedback control loop. Furthermore, we developed a generic dependency model which captures dependencies among DBMS' sensors and effectors and can be used in the process of autonomic database performance tuning. Knowledge is needed for automating typical tuning tasks like predicting performance problems, finding their root causes and finally resolving these problems. Figure 6 sketches the application of Knowledge Model components in the according phases of MAPE.

<table>
<thead>
<tr>
<th>MAPE Phase</th>
<th>Referenced Knowledge Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monitor</td>
<td>Sensor Knowledge, Sensor History, Workload Knowledge, Policy Knowledge</td>
</tr>
<tr>
<td>Analyze</td>
<td>Events, Event Correlation Knowledge, Event History, Workload Knowledge, Policy Knowledge</td>
</tr>
<tr>
<td>Plan</td>
<td>Statistics, Dependency Knowledge, Tuning Plans, Event Tuning Plan Subscription, Sensor History, Tuning Plan History, Cost Knowledge, Decision Logic, Workload Knowledge, Policy Knowledge</td>
</tr>
<tr>
<td>Execute</td>
<td>Tuning Plans, Tuning Plan History</td>
</tr>
</tbody>
</table>

Figure 6. MAPE phases with corresponding knowledge components

Unfortunately, obtaining machine-interpretable database tuning best-practices is not easy. The information provided on the internet is unstructured and formalizing tuning best-practices manually is a cumbersome job. Thus, our near-term research focus is to establish a community platform for sharing database tuning best-practices and to provide tooling support for DBAs.

Furthermore, in order to facilitate the DBAs to specify the high-level policies and to let the system tune itself concepts to map high-level to low-level policies and low-level policies to events need to be developed. Additionally, further research work has to delve into the influence of policies among each other and into the impact of multiple (possibly conflicting) policies on the performance optimization. This requires having a sophisticated policy and event model.

Much work is also needed to understand, characterize and predict workloads and interpret their impact on other components and interaction between components.

The research field of autonomic database performance tuning is very promising. However, we do not believe that highly skilled DBAs will ever get replaced by intelligent autonomic database administrating tools. Automation is in fact a great option for the “usual case” but there always will be exceptional cases that need to be taken care of.

REFERENCES


134