Fusion Engineering and Design 129 (2018) 12–15

Contents lists available at ScienceDirect

Fusion Engineering and Design

journal homepage: www.elsevier.com/locate/fusengdes

TokSearch: A search engine for fusion experimental data

B.S. Sammuli*, J.L. Barr, N.W. Eidietis, K.E.J. Olofsson, S.M. Flanagan, M. Kostuk, D.A. Humphreys

General Atomics, 3550 General Atomics Ct, San Diego, CA, USA

ARTICLE INFO

Keywords:
Apache spark
Big data
Data archive
Parallel processing

ABSTRACT

At a typical fusion research site, experimental data is stored using archive technologies that deal with each discharge as an independent set of data. These technologies (e.g. MDSplus or HDF5) are typically supplemented with a database that aggregates metadata for multiple shots to allow for efficient querying of certain predefined quantities. Often, however, a researcher will need to extract information from the archives, possibly for many shots, that is not available in the metadata store or otherwise indexed for quick retrieval. To address this need, a new search tool called TokSearch has been added to the General Atomics TokSys control design and analysis suite. This tool provides the ability to rapidly perform arbitrary, parallelized queries of archived tokamak shot data (both raw and analyzed) over large numbers of shots. The TokSearch query API borrows concepts from SQL, and users can choose to implement queries in either Matlab™ or Python.

1. Introduction

Fusion experiment archival systems collect and store large volumes of data, both on a per shot basis and in aggregate across many shots. DIII-D, for example, has run over 170,000 shots, each of which, in recent years, has consisted of tens of gigabytes of data. A common research workflow is to perform detailed analysis of a particular shot using typically 10–100 signals. However, if analysis is needed on data spanning many shots, such as for machine learning, the throughput achieved by retrieving and processing data serially may prove to be untenably slow. When performing such analyses, it quickly becomes apparent that a principled approach to parallelizing both data retrieval and processing is needed. TokSearch, part of the General Atomics TokSys control design and analysis suite [1], aims to provide a convenient, highly parallelized environment in which researchers can search for relevant data and apply arbitrary analysis to it.

2. Specifications

The primary goal for TokSearch is to enable the traversal of tens to hundreds of thousands of shots, sufficiently rapidly and efficiently to support large-scale database studies developed on demand. This general goal and the specific use cases envisioned impose many desirable and required design features, including interface characteristics, speed of execution, deployment, data handling, and programming language selections. The resulting requirements are described in more detail in the sections following.

2.1. Result types

TokSearch was designed with three primary use cases in mind:

1. Perform a query that returns a result set that fits in memory on the client host.
2. Perform a query that generates a large data set and then persists the result to disk.
3. Perform a query and cache the results in cluster memory for subsequent analysis.

2.2. Programming language options

In an effort to provide as convenient an environment as possible, TokSearch was designed to support two languages that are commonly used in the fusion community: Matlab and Python. Although the TokSearch API borrows concepts from SQL, all queries are constructed entirely in one of these two languages and then executed either in batch mode or from an interactive REPL (read-eval-print-loop) environment.

2.3. Deployment options

TokSearch works best on a dedicated cluster with distributed storage. However, TokSearch was designed to be able to deploy on a single

* Corresponding author.
E-mail address: sammuli@fusion.gat.com (B.S. Sammuli).

https://doi.org/10.1016/j.fusengdes.2018.02.003
Received 9 June 2017; Received in revised form 31 January 2018; Accepted 1 February 2018
0920-3796/ © 2018 Published by Elsevier B.V.

host and take advantage of multiple CPU cores on that host. This way, TokSearch can be utilized without the need for a multi-computer cluster.

2.4. Data sources

TokSearch is designed to abstract various types of data sources using a common interface. As of this writing, the TokSearch supports data retrieval from:

1. PTDATA – a DIII-D-specific archive format
2. MDSplus – both thin and thick clients
3. Parquet files – See discussion in Section 4.2

Support for HDF5 (Hierarchical Data Format) is planned as a future upgrade. This would enable TokSearch to interact with the ITER CODAC archiving format [2].

2.5. Flexibility and scalability

TokSearch is designed to flexibly interchange its underlying parallelization technology. Currently the Python API supports Apache Spark™ [3], which is a cluster-computing framework that runs on commodity hardware. The Matlab API supports use of both the Matlab Parallel/Distributed Computing toolbox and Spark.

Another goal for TokSearch is to achieve horizontal scaling. That is to say, to increase the shot-processing rate as a linear function of the number of cluster nodes (or cores). We provide a demonstration of this later during the discussion of performance results.

3. Search semantics

TokSearch represents the search domain of a query as a table (Fig. 1). This table is similar to a view in a traditional relational database in that the data for each column may be physically stored in disparate locations and is only materialized upon query execution. Each row in the table contains the data for a single shot (or shot segment), and each column represents a signal. The data type of the columns is arbitrary, but will typically consist of a time series structure with data and times fields.

Each signal need not be stored in the same location; the TokSearch API allows for signals from multiple sources to be fetched during a single query.

Fig. 2 illustrates at a high level the per-row algorithm executed by TokSearch for each shot.

1. For a row *r* each signal s_i is fetched from a data source.
2. *r* is processed such that a set of derived quantities are appended to it, resulting in a new row *R* (Fig. 3). Appending intermediate results at this stage allows multiple subsequent calculations to make use of the derived quantities without recalculation.
3. A user-defined filter w = w(*R*), analogous to the where clause in an SQL query, is applied to *R*.
4. If the row matches, that is, if w(R) returns a Boolean TRUE, then the query appends data to the result set. The fields C = $\{c_1,\dots, c_L\}$ in the result set are defined as a list of either strings specifying fields in R, or as functions that perform additional user-defined transformations. In other words, each field c_j in C can be one of the following:

shot	s_1	…	s_n
1	[times, data]	…	[times, data]
…	[times, data]	…	[times, data]
M	[times, data]	…	[times, data]

Fig. 1. An illustration of a TokSearch table. The query author provides a list of n signals {s1, …, sn} which are represented as columns, and M shots.

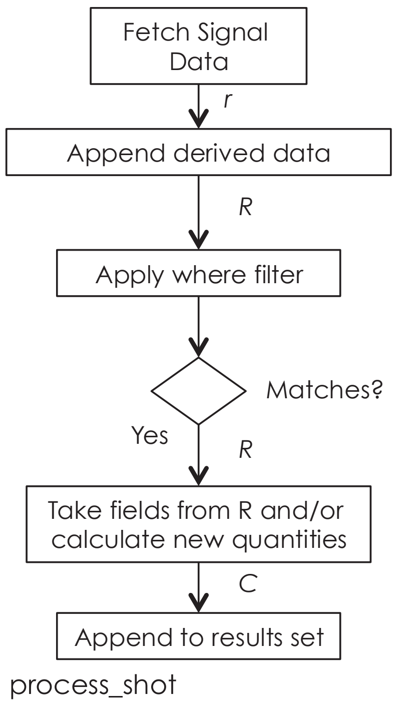

Fig. 2. High level overview of the TokSearch query execution sequence applied to each shot.

Function d_j	Transformed Row
---	$r = \rho_0 = \{s_1, \dots, s_n\}$
d_1	$\rho_1 = \{s_1, \dots, s_n, d_1(\rho_0)\}$
d_2	$\rho_2 = \{s_1, \dots, s_n, d_1(\rho_0), d_2(\rho_1)\}$
…	…
d_N	$R = \rho_N = \{s_1, \dots, s_n, d_1(\rho_0), \dots, d_N(\rho_{N-1})\}$

Fig. 3. Given a list of functions $\{d_1,\dots, d_N\}$, during query execution each d_j will be applied sequentially, appending the result onto an intermediate row ρ_j. The previous intermediate row in the transformation, ρ_{j-1}, is available to d_j. The resulting row R is then available to all subsequent query execution steps.

- A signal from the table with, for example, sub-fields times and data;
- A derived value (as calculated in step 2 above);
- A value returned by applying a user-supplied function to R.

Using the notation defined above, we can now express a TokSearch query in pseudo-SQL:

```
SELECT c_1, …, c_L
FROM R
WHERE w(R) = = true AND shot in shots;
```

4. Parallelization and data locality

TokSearch queries can be run in a single thread on a single host, as shown in Fig. 4 below. This is often useful for debugging a query on a small subset of the total search domain.

TokSearch executes in parallel by simply running multiple instances of the single-threaded scan, each with a partitioned subset of the total shot list. Fig. 5 illustrates this.

As discussed in Section 2.5, TokSearch queries will run on multiple types of clustering environments. In all cases the basic architecture is to have a master process that interacts with one or more parallel workers, each of which may be distributed across multiple host computers.

4.1. Network-based data retrieval

The simplest means of parallelizing data retrieval is for each worker