Stream Processing Based Intelligent Transport Systems

Eric Bouillet, Mark Feblowitz, Zhen Liu, Anand Ranganathan, Anton Riabov, Schuman Shao, Don Schlosnagle, Fan Ye

IBM T. J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532, USA
Email: {ericbou, mfeb, zhenl, arangana, riabov, smshao, daschlos, fanye}@us.ibm.com

Abstract—
In this paper, we present a Fleet Management Center application implemented using a stream processing infrastructure we call System S. System S enables the deployment of large scale applications with mechanisms for sharing multi-party data sources, software components, and even intermediate results. This approach significantly reduces the cost of software integration, and ownership, the major factor in Intelligent Transportation Systems. In addition, the system includes an adaptive data source management that determines the list of relevant data sources based on the current locations of the entities monitored or managed by the applications.

I. Introduction

Intelligence Transportation System (ITS) applications are critical to improve the efficiency of modern transportation [7]. Some ITS applications are capable of answering complex queries such as finding the fastest routes for individual vehicles taking into account realtime road conditions [6]. Building a scalable and flexible ITS system, however can be a tremendous challenge. Traffic data come from many heterogeneous sources as streams, in different forms, content and quality. Both spatial and temporal analysis is needed to correlate data from sources in large geographic areas or time periods. The demands of analytical results come from many user groups, such as drivers, highway patrol, department of transportation. They pose not only large numbers of simultaneous queries, but also queries of significantly different nature.

Not surprisingly, software development and integration is a major cost factor in ITS applications. It is important that we make these software artifacts as *reusable* and *modular* as possible. We must be able to compose them effectively in different applications that can answer the various kinds of queries. Such a system requires standard and reusable mechanisms for interfacing the data with the outside world. In order to facilitate growth and incorporation of new technologies, the system must enable the creation and deployment of new ITS applications without disrupting existing ones. Furthermore, new applications should be able to reuse intermediate results produced by existing applications in order to minimize duplicate or redundant processing of data. Finally, the system must allow location-sensitive applications to adapt to the current locations of the entities they are tracking or assisting.

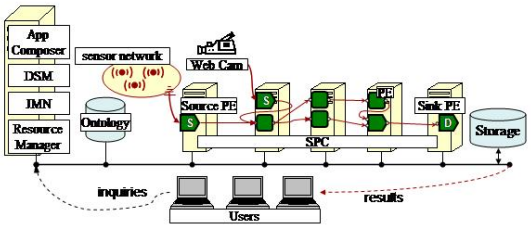

Fig. 1. An application that takes in data streams from sensor networks and webcams is deployed in System S.

In this paper we describe a Fleet Management Center (FMC) ITS framework, and show how we address those challenges using many interconnected modular, reusable software components called *Processing Elements* (PEs), each of which takes data of certain content and format and perform various kinds of analysis. Different PEs may be interconnected in a processing graph (or workflow) that takes in data from different sources and produces desired end results (such as directions to a fleet of vehicles along the shortest routes using real time traffic conditions). The PEs are deployed in System S [3], an infrastructure that supports the deployment of large scale stream processing applications on the fly. We also use a Data Source Manager (DSM) to dynamically select data sources that are relevant to our ITS application.

The rest of the paper is organized as follows. We give an overview of System S in Section II. Section III explains how application adaptation and data source management are accomplished in the system. We illustrate a fleet management application and provide vehicle routing performance results in IV. We conclude in Section V.

II. System Overview and Examples

We have implemented our approach within a stream processing system referred to as System S. A sample configuration of System S with a deployed application is shown on Figure 1. The system is centered around a *Stream Processing Core* (SPC) [3], a scalable distributed middleware for stream processing. The SPC consists of a large cluster of dedicated processors and an execution context upon which stream processing

1-4244-1178-5/07/$25.00 ©2007 IEEE.

applications are deployed. Each stream processing application consists of a processing graph, which is a directed acyclic graph of stream processing components, namely data sources and processing elements (PEs).

Data sources deliver streaming *primal data* into the system and PEs perform various types of operations on streaming data - filtration, annotation, transformation, stream join, etc. PEs are individually deployable and reusable software components and are interconnected by multi-access (single-writer, multiple-reader) streams.

End-users interact with System S by specifying stream queries that describe the desired end results to them. We refer to System S queries using the name *inquiries*. Inquiries are pursued by assembling and deploying processing graphs of interconnected PEs that can produce the desired end results. These processing graphs may be assembled manually or automatically. In this paper, we demonstrate System S's ability to support Intelligent Transport System oriented applications using processing graphs that have been assembled manually by an application developer or administrator. These graphs can be parameterized for different use-cases. In [1] we describe an ontology-based application composer for System S that automatically composes the processing graph for an inquiry using detailed descriptions of data sources and PEs in ontologies along with AI planning algorithms. We do not elaborate on the details of automatic composition here.

For purposes of illustration, consider as a simplified example: an inquiry requesting traffic congestion reports for a particular roadway intersection. Such an inquiry may draw audio data from a sound sensor data source and apply an Audio Pattern Analysis PE, which matches the data to known audio patterns to determine the level of congestion at the intersection (isolating the lower thread in Figure 2). In order to improve the accuracy of such an assessment, the application may use a Video source, extracting images from the video stream and examining them for alignment to visual patterns of congestion at an intersection (the upper thread). The end result would then be achieved by joining feeds from the two analytic chains.

This simple workflow can be extended further, by including additional PEs and data sources. In fact, it describes a small part of the Fleet Management Center service (FMC) application discussed in Section IV. In addition, applications in FMC provide vehicle routing services based on the analysis of real-time data obtained from the sensors.

Each data source delivers primal data into System S by way of a *SourcePE*. The SourcePE (denoted by "S" in Figure 1), a special type of PE that also runs in the SPC, serves the sole function of bringing data from an external source into the system. SourcePEs encapsulate the drivers, and other source dependent configuration setup to access a broad variety of external sources.

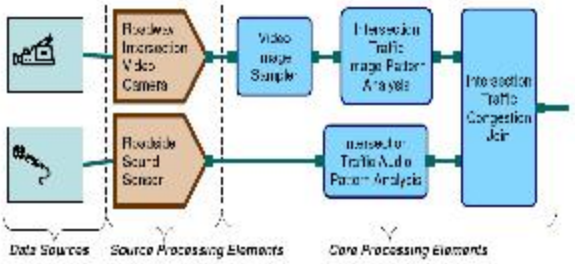

Fig. 2. A stream processing graph that utilizes roadside cameras and microphones to produce traffic congestion information.

The data is packaged into internally streamable and processable *Stream Data Objects (SDOs)*. SDOs are lightweight data containers that carry data from SourcePEs to PEs, from PEs to other PEs, and eventually to SinkPEs (the dual of SourcePEs, denoted by "D" in Figure 1) that deliver inquiry result SDOs outside of the SPC (e.g., to an end user or a data store). Source PEs and Sink PEs allow the mechanisms for interfacing with the outside world to be reusable. The same Source and Sink PEs can be reused with different parameters for bringing in data from similar data sources and for pushing out results in similar manners.

PEs and data stream instances in SPC are statically assigned individual security and privacy labels, inherited in part from the applications' effective user IDs, and the PE classes. SPC connects PEs to the data streams only if their respective security and privacy labels allow it. If necessary the labels can be adjusted using "downgrader" PEs that sanitize sensitive information (e.g. remove identifying information, etc). Applications can tap into past (archived) or present (real-time) data. Past data is stored by the system and can be extracted and processed as a stream.

PEs can be co-deployed on the same processing node (hardware) and can also be replicated across multiple nodes. Data can be processed serially, from one PE to another (linear graphs), but parallelizable tasks can be performed more efficiently by streaming data to concurrently operating distributed PEs. The PE lifecycle management and resource allocation tasks are carried out by a Job Manager (JMN), an integral system component of System S. The JMN receives requests to deploy PEs with their flowgraph specifications represented in a *Job Description Language* (JDL). The JMN interprets the JDL, computes the adequate resource allocation with respect to resource availability, and QoS constraints and priorities specified in the JDL, and deploys the PEs in SPC. In addition it provides APIs to monitor and terminate deployed PEs.

Another component of System S is the *Data Source Manager* (DSM) which manages and activates data sources based on requests and feedbacks from applications and users. The DSM provides a virtualization of the data source from the application perspective, allowing application developer (or automatic com-

poser) to focus on instrumenting the data analytics independently of how the data is brought into the system.

An important feature of our infrastructure is the ability to reuse intermediate streams across different applications. Once an application (or processing graph) is built and deployed in response to a users inquiry, the set of intermediate streams produced in this processing graph are added to the set of data sources that are managed by the DSM component. These intermediate streams are called derived streams. Subsequent applications that are deployed can make use of these derived streams. The main advantage of this feature is to allow sharing of common processing across applications and to avoid unnecessary duplicate computation.

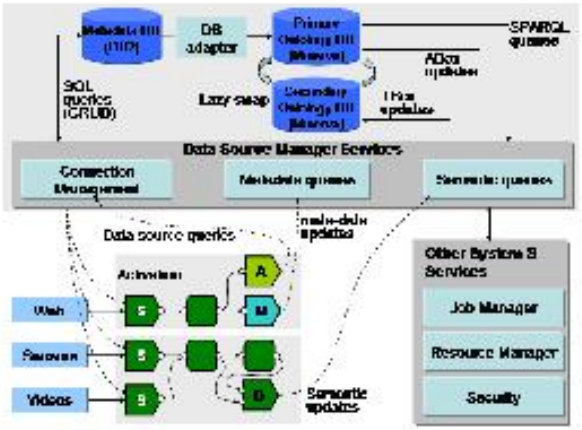

Fig. 3. Data Source Management Service Architecture of System S

III. Data Source Management

The Data Source Manager (DSM) component provides a repository with services for managing external data sources that are available to the system. DSM services include: (1) managing information about all sources, (2) providing the appropriate source PE configuration to connect to external sources, and (3) activating or deactivating sources PEs depending on current application needs. DSM also supports data source specific monitoring services, implemented in System S as stream processing applications that monitor the availability and quality of existing sources.

As illustrated in Figure 3, DSM consists of a meta-data and a semantic (Minerva) database, and services to interact with these databases. The meta-data database contains the system attributes of the data source, such as type of data produced by the source, the URL and configuration parameters to access the data source. It also contains real time attributes such as data rates and packet loss ratios, which can be used by other system components to rank the quality and manage active data sources. The semantic database extends the meta-data with attribute relationships and additional descriptions of the data sources expressed as facts in the Web Ontology Language (OWL [4]). In addition, it supports reasoning based on Description Logic Programs (DLP [2]). It supports a SPARQL [5] query interface and provides results to queries, that include both directly asserted facts as well as facts that are inferred using DLP reasoning. A key characteristic of the semantic database [8] is that it can perform incremental updates to the ABox (assertions on instances) of the ontology, but any updates to the TBox (assertions on concepts) requires a regeneration of the results of reasoning and is time-consuming. Hence, any update to the ABox (such as the current state or quality of a data source) can go directly to the semantic database. However, even this operation is slower compared to a traditional database due to the necessity of checking the logical consistency of the new facts. Thus, if the update rate of the metadata of a data source is very high, the updates are first performed in the meta-data database, where they are aggregated and periodically propagated to the semantic database by the DSM database adapter. When the TBox changes (which do not occur very often in our system), the changes are made to a standby semantic database which regenerates the results of reasoning. When the standby database is ready, the DSM swaps the two databases in order to make the TBox changes available to the system.

Stream application composers (human or automated) express their requirements for a data source in the form of RDF semantic queries using the SPARQL language. DSM executes the query on its semantic database to determine the list of potential data source instances suitable for the application. Data sources selected by the stream application composers are then connected with Source PEs that are configured according to the configuration attributes specified in the meta-data database of DSM.

After deployment the various applications can refine the desired data source semantic descriptions, e.g. limit their interest to sensors located in certain areas if all the vehicles of interest are concentrated in that area. This is achieved using application specific DSM agents (PE 'A' in Figure 3) that translate the data source requirements into the corresponding SPARQL queries. The DSM Connection Management service uses the SPARQL queries to maintain a list of data sources of interest, and activate or deactivate the corresponding source PEs.

IV. Application to Fleet Management Services

A. FMC Application Architecture

In FMC, routes for delivery trucks are set at the beginning of the shift, informed by any known traffic conditions at the time of departure. There is a standing inquiry for each truck; it watches for changes in traffic conditions that warrant route replanning and provide an updated route to the truck. The routes are based on a collection of destinations and the current location

of the vehicle (determined by a GPS transmitter in the vehicle). Accompanying the standing, vehicle-specific routing inquiries are inquiries that examine streaming data from multiple sensors and other sources and update the roadway and traffic conditions, with special focus on corridors covering known vehicle destinations.

The stream processing application is illustrated in Figure 4. The application is built using the System S stream processing infrastructure, and consists of a set of PEs arranged into processing graphs. The upper portion of the processing graph depicts the route update, and the lower portion depicts the location condition update. In the route update portion are the various PEs - analytic modules that receive the streaming data and perform functions such as generating the K best potential travel corridors (a corridor delineates a region set of the map that is likely to be traversed by the vehicle), deciding on routes based on vehicle types and, where available, traffic conditions. The two main results of this portion of the application are the route updates for the vehicles and updates to the list of currently active locations. The results guide the focus of the condition-assessment processing (in the lower part of the graph). The route update portion involves the following PEs:

Vehicle LOC: Receives GPS data of current vehicle locations and package it into internally streamable and processable SDOs.

Vehicle DEST: Retrieves a list of destination coordinates for the vehicle, and package it into SDOs. Various implementations of the PE can either retrieve this information from a database, or receive it directly from user console.

Join Vehicle ID: Joins the LOC and the respective DEST coordinates of every vehicle received on its input ports. It generates a new join SDO, containing the vehicle ID, its current location, and the list of stop coordinates, when this information changes.

Potential Corridor Generation: Generates the corridors that might be traversed by the vehicle. The output SDO is the input augmented with the list of regions contained in the corridor (e.g. states or zip codes), and way-points (main cities, etc). The set of way-points constitutes a chain of *virtual* links, for each of which a route must be computed. For scalability reasons, this PE uses a summarized road-map information, which is not frequently updated.

Locations of Interest: Store the corridors in the location list database for use by other PEs in the condition assessment applications.

Vehicle Route Decider: Several PEs, assigned to a particular region (e.g. one PE per state). Each PE listens for SDOs that contain a corridor intersecting with its region (other SDOs are ignored). The PE looks in those SDOs for or virtual links that are located within its assigned region and replace them with the shortest routes.

Join Route: This PE merge results from the various Vehicle Route Decider (excluding virtual links) into a subgraph. When it can find a shortest-path in the subgraph, it generates an SDO with that route.

The lower portion contains an inquiry per known data source (weather sensor networks in this example), drawing data from the source, processing the data to determine its location and the conditions, and updating a Location Conditions store, the sole recipient of condition inquiries' result data. This data is retained for some limited duration and triggers rerouting in the upper inquiries. Other data sources might include non-sensor data sources such as local radio weather report sources or traffic incident report sources. But these are secondary sources, drawn from other service providers, possibly providing stale reports.

For rapidly changing conditions, it would be better to draw data from directly accessible sensors or sensor networks, using devices such as traffic surveillance cameras, in-road vehicle sensors, wireless sensors capable of sensing temperature, barometric pressure and humidity, infrared remote temperature detectors (to detect road surface temperatures), and even roadside microphones (to analyze traffic noises). In more permanent installations where power consumption is not a factor, each of these can produce high volumes of streaming data, which can be processed by many stream processing applications into usable results.

There are a few non-sensor sources, providing information not limited to a single geographic area; these may deliver weather updates that span a wide range. There are many more sensors, though, since the scope of the data is limited by the sensor's range: a traffic camera at street intersection can only deliver data within the camera's (possibly fixed) range. Many sensors covering anticipated travel corridors must be be deployed, and activated by DSM when respective travel corridors are active, with their data processed by separate stream-interconnected PEs, in response to inquiries. The location condition update consists of the following PEs:

DSM Agent: translates the location list into semantic queries for the DSM connection management which in turn activates the appropriate source PEs to satisfy the requests. There is one specialized DSM agent per modality, e.g. weather, traffic report.

Sensor Network Gateway PE: is a source PE specialized to query sensor gateways. DSM connection management dynamically reallocate source PEs in response to requests from the DSM agent.

Location Extractor and Loop Data Congestion Analytics: are one of many PEs used to assess the road conditions. In this example they extract the Induction Loop data, and analyze its content to estimate the impact on traffic delays.

Condition update: persist weather related location condition to a Road Condition database.

The application also includes sink PEs (map vehicle position, view route, and map condition) that present the results of the user inquiry to a user interface,

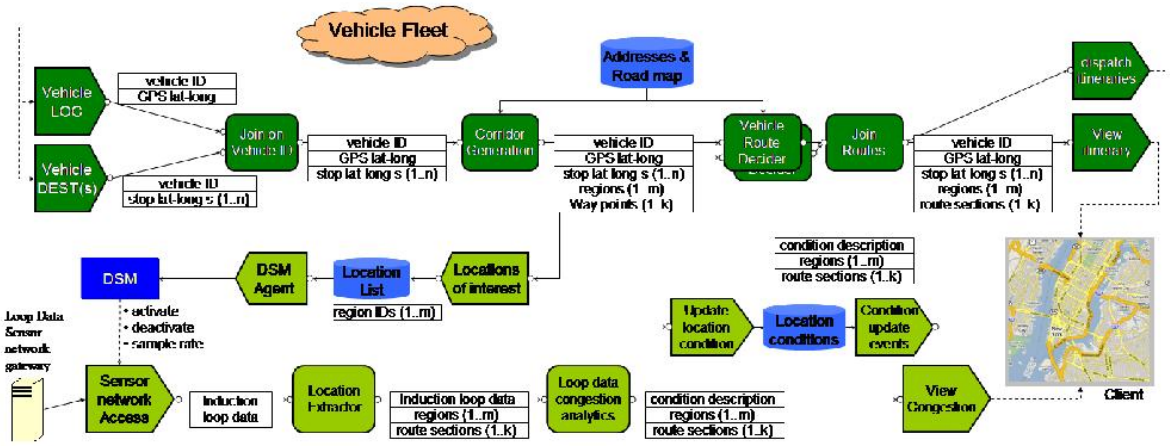

Fig. 4. Fleet Management Center Application. The application consists of route computation PEs (top portion of the figure), and an example of route condition-assessment PEs (bottom part).

or automatically send that information directly to the vehicles.

B. Incremental Application Deployment and Derived Stream Reuse

Note that the route assessment application can easily be extended with new applications to increase the accuracy of the route report, as new modalities (or processing resources) become available. This is because the location list and the location condition databases provide a separation between the PEs that update the database, and PEs that read their data. This separation allows the PE in the top part of Figure 4, which update the corridors, to be developed and deployed independently of the PEs in the lower part, which are responsible for updating the location conditions. For instance several route selection PEs can be deployed in parallel to handle fleets of vehicles that have different routing requirements. Similarly, diverse applications can be deployed to update the location conditions with new modalities that corroborate, or improve the accuracy of the location condition.

Derived data streams from existing applications can also be registered in DSM as a Data Source, and reused to support other applications. Reuse is allowed within the limits of the respective application security and privacy constraints, which are enforced by SPC. For instance if the vehicles managed by the FMC application are shuttle vans, we can extend the application to dispatch vans to pick new customers along their assigned route. In this example, the application uses a source PE that takes customer pickup and drop-off street addresses in input, a PE that converts the addresses in lat-long coordinates, and a PE that matches the coordinates to the routes (derived stream of the Join Route PE in the original FMC application), currently assigned to the vans. The later PE selects the van that is the best match for this customer and modifies its route accordingly to include the customer.

C. Experiments

In this experiment, we measure the performances of the route update, the upper portion of the processing graph in Figure 4, which is the most CPU intensive part of the FMC application. For the purpose of the experiment, we randomly generate a global map of $10,000$ nodes connected by $40,000$ road-sections that are $1km$ to $10km$ long. Nodes in the global map are uniformly distributed into four regions (A,B,C,D), such that 1% of the road-sections connect nodes located in separate regions (A-B, B-C, or C-D). We decompose this global map into five maps: (1) a corridor map consisting of the road-sections crossing a region border and the nodes that they connect to (a virtual connection is assumed from those nodes to all other nodes located in the same region) ; and (2) four individual road maps, one per region, containing the nodes within that region, as well as nodes in other regions that are directly connected by a road-section to a node in the region, and the road-sections between the nodes. As illustrated in Figure 5, the corridor map is used by the *Corridor Generation* PE, and the regional maps are used by the *Vehicle Route Decider* PEs of the distributed routing.

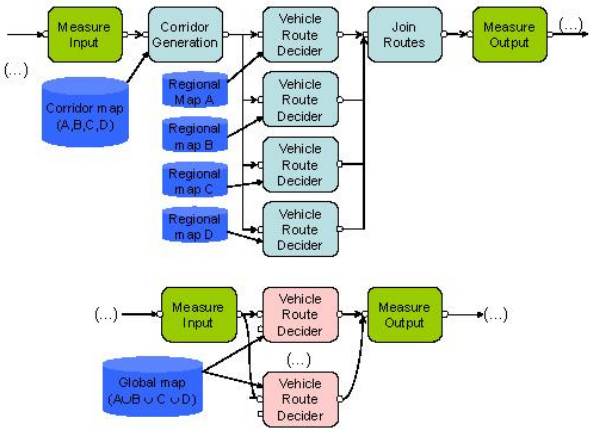

Fig. 5. Experiment setup. The top part of the figure shows the distributed workflow of the routing algorithm. The bottom part shows the centralized routing workflow used for comparison.

We compare the distributed route update with a centralized Dijkstra shortest-path (also a *Vehicle Route Decider* PE), using the global road-map. The workflow of the centralized routing is shown in the lower part of Figure 5. In order to provide a fair comparison with the distributed route update, we uniformly balance the routing queries of the centralized routing between $N = 5$ instances of the *Vehicle Route Decider*, so that the same amount of CPU resources is available to both implementations. We also include the performances of a centralized approach without load balancing using a single *Vehicle Route Decider*.

We approximate the application as a queuing system. In such a system, the total run time T consists of a constant delay t, plus a processing time spent in the most CPU intensive PE. This processing time depends on the maximum number C of queries per seconds that the system can handle, and varies with the query arrival rate λ. Other processing time spent in less CPU intensive PEs are comparatively negligible, and included in t. The total run time is equal to $T = t + 1/(C - \lambda)$. If we vary the arrival rate λ, and measure the corresponding runtime T, we can thus derive the respective values of C and t.

As depicted in Figure 5, we insert two monitoring PEs to carry out the measurements. The *Measure Input* PE measures a moving average of the inter-arrival rates λ, and annotates the input SDO with a timestamp. The *Measure Output* PE uses this timestamp information (which is transparently carried through the intermediate PEs), to compute the total runtime T. The two PEs are collocated on the same host, so that the same clock reference is used to compute the intervals. All other PEs are allocated on separates hosts in order to distribute the CPU utilization. The target deployment testbed used in our experiments consists of four 4-way 3GHz Intel Xeon(TM) nodes and five 2-way 2.4GHz AMD Opteron (TM) 250, running the Linux Suse 9.3 operating system and interconnected with 1Gbs network cards via a Cisco Catalyst 6509 switch. For all the experiments, we use the same sequence of routing requests, which consist of randomly generated source-destination pairs uniformly distributed across the map.

Figure 6 shows the results of our experiments. The corresponding values of C and t are shown in Table I. The table also include the average route lengths. The results indicate that the distributed routing offers 74% more throughput than the centralized approach with load balancing ($N = 5$ CPUs). However, the distributed approach computes routes that are 7% longer than the centralized approach on the average. This penalty is due to the lower level of details available in the corridor map.

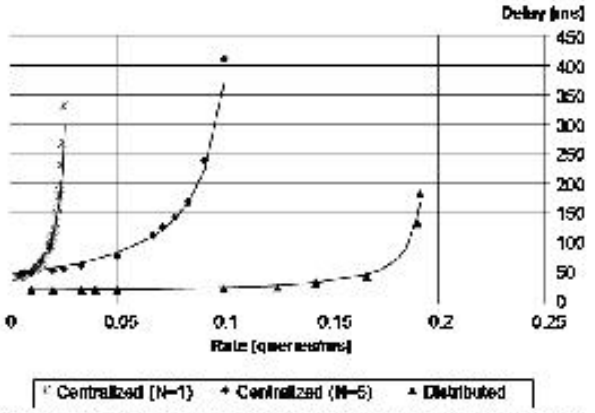

Fig. 6. Delays of the distributed versus the non-distributed stream-based route determination architecture.

TABLE I

PERFORMANCES OF THE ROUTING ARCHITECTURES.

Implementation	C (queries/s)	t (ms)	Avg length
Distributed	198	10	169.59km
Centralized (N=5)	114	0	158.79km
Centralized (N=1)	29	0	158.79km

V. CONCLUSION

In this paper we describe a Fleet Management Center application implemented using System S, a distributed infrastructure for deploying large-scale stream processing applications. The use of a stream processing oriented architecture addresses several of the challenges faced in the implementation of ITS applications. It promotes the paradigm for reusable and modular software components that can be composed in different applications to answer different kinds of ITS queries; it provides reusable components for interfacing the data with the outside world; and it provides the mechanisms for managing concurrent applications, share derived streams across applications, and extend existing applications into larger applications without disturbing the existing ones.

REFERENCES

[1] E. Bouillet, M. Feblowitz, Z. Liu, A. Ranganathan, A. Riabov, and F. Ye. A semantics-based middleware for utilizing heterogeneous sensor networks. In *Int' Conference on Distributed Computing in Sensor Systems*, 2007.

[2] B. Grosof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: combining logic programs with description logic. In *WWW'03*, pages 48–57, 2003.

[3] N. Jain, L. Amini, H. Andrade, R. King, Y. Park, P. Selo, and C. Venkatramani. Design, implementation, and evaluation of the linear road benchmark on the stream processing core. In *SIGMOD'06*, June 2006.

[4] D. McGuinness and F. van Harmelen. Owl web ontology language overview. In *W3C Recommendation*, 2004.

[5] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. In *W3C Working Draft*, 2006.

[6] K. Seongroon, M. Lewis, and C. White. Optimal vehicle routing with real-time traffic information. *IEEE Transactions on Intelligent Transportation Systems*, 6:178– 188, june 2005.

[7] US Department of Transportation. Intelligent transport services. *http://www.its.dot.gov*.

[8] J. Zhou, L. Ma, Q. Liu, L. Zhang, Y. Yu, and Y. Pan. Minerva: A scalable OWL ontology storage and inference system. In *The 1st Asian Semantic Web Symp.*, 2004.