DEMO: Integrating MPC in Big Data Workflows

Nikolaj Volgushev
Boston University
nikolaj@bu.edu

Malte Schwarzkopf
MIT CSAIL
malte@csail.mit.edu

Andrei Lapets
Boston University
lapets@bu.edu

Mayank Varia
Boston University
varia@bu.edu

Azer Bestavros
Boston University
best@bu.edu

ABSTRACT

Secure multi-party computation (MPC) allows multiple parties to perform a joint computation without disclosing their private inputs. Many real-world joint computation use cases, however, involve data analyses on very large data sets, and are implemented by software engineers who lack MPC knowledge. Moreover, the collaborating parties – e.g., several companies – often deploy different data analytics stacks internally. These restrictions hamper the real-world usability of MPC. To address these challenges, we combine existing MPC frameworks with data-parallel analytics frameworks by extending the Musketeer big data workflow manager [4]. Musketeer automatically generates code for both the sensitive parts of a workflow, which are executed in MPC, and the remaining portions of the computation, which run on scalable, widely-deployed analytics systems. In a prototype use case, we compute the Herfindahl-Hirschman Index (HHI), an index of market concentration used in antitrust regulation, on an aggregate 156 GB of taxi trip data over five transportation companies. Our implementation computes the HHI in about 20 minutes using a combination of Hadoop and VIFF [1], while even "mixed mode" MPC with VIFF alone would have taken many hours. Finally, we discuss future research questions that we seek to address using our approach.

1. INTRODUCTION

Big data analytics are a key part of modern business processes. Companies and regulatory agencies can draw vital insights from running such analytics, especially when they are executed across data sets from multiple sources. However, the justified privacy concerns related to proprietary data sets are a major hurdle to running such computations across multiple competing organizations, even if knowing the result serves a common interest.

Secure multi-party computation (MPC) is a cryptographic technique that allows independent parties to jointly compute a shared result without revealing their private inputs to the computation. MPC has been an active area of cryptography research since the 1980s [15], and recent advances focus on applied aspects of MPC; MPC frameworks such as VIFF [1], Sharemind [3], and Wysteria [10] allow end-users to run arbitrary programs in MPC, as long as they are completely re-implemented in the chosen framework's front-end language.

However, many real-world use-cases only necessitate MPC for a few crucial operations as part of a larger workflow. While some MPC frameworks support a "mixed-mode" operation that combines local computation with secure, distributed MPC steps [10],[1] real-world use of MPC currently still faces three key challenges:

1. MPC integrates poorly with existing analytics workflows and widely-used data processing systems;

2. Significant expert knowledge is required to implement and run analytics in an MPC framework; and

3. MPC frameworks scale poorly to large data sets, since they do not support efficient data-parallel processing outside MPC.

In this work, we address these three challenges. By doing so, we demonstrate that use of MPC can be made viable for societally important use cases that involve large data sets, such as bank stress tests and early detection of market oligopolies (§2).

To make MPC more accessible to non-experts and industry data analysts, we have added support for secure multi-party computation to the Musketeer big data workflow manager [4]. Musketeer automatically generates code for a variety of data processing frameworks (such as Hadoop, Spark, and Naiad) from a high-level workflow description (e.g., a SQL language). With our extensions, Musketeer generates MPC code automatically from programs specified in a relational language inspired by LINQ [8], requiring no expert knowledge. It also automatically embeds the MPC into larger workflows that involve private processing steps on multiple organizations' heterogeneous data analytics clusters. Even if organizations use different data processing stacks, we automatically generate both the preprocessing code and the "glue code" for embedding MPC in the workflow. Specifically, our contributions are:

1. Proof-of-concept integration of MPC into typical "big data" workflows specified in a high-level relational language (§3).

2. The extension of the Musketeer workflow manager with automatic code generation for secure MPC steps (§4).

3. Implementation of an example use case that highlights the advantages of our approach: a market share computation in which private sales records are preprocessed to compute the Herfindahl-Hirschman index in "mixed-mode" MPC (§5).

CCS'16 October 24-28, 2016, Vienna, Austria
© 2016 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-4139-4/16/10.
DOI: http://dx.doi.org/10.1145/2976749.2989034

[1] The "mixed-mode" term is overloaded: sometimes, it is used to mean a combination of different *types* of MPC (e.g., arithmetic MPC based on secret sharing [12] and boolean MPC based on garbled circuits [15]). Our system can also support the latter (§7).

Our evaluation compares the runtime of our market share computation to two extremes: first, not using MPC and allowing a trusted third party (e.g., a regulator) to run the computation; and second, running the entire computation in an MPC framework. We find that our integrated workflow executes almost as fast as the insecure baseline while requiring no trusted third party, and that it runs substantially faster than when using an MPC framework only.

2. EXAMPLE USE CASES

Having MPC as part of a data analytics workflow enables numerous use cases. Many examples involve computations across business competitors, either because the aggregate result is of interest to all of them, or because a regulating authority has an interest in monitoring the market. We discuss two concrete examples below.

Bank stress tests. In the wake of the global financial crisis of 2008-09, financial regulators have devised metrics to measure systemic market risk. Currently, such stress tests are laboriously executed, with each bank manually aggregating data for submission to the regulator and covering only a part of its assets and investments.

Instead, banks could use MPC to jointly run continuous stress tests on their respective books in their entirety by integrating MPC into their existing data analytics stacks. Abbe *et al.* [2] suggested the use of MPC for this problem, while Narayan *et al.* [9] model the computation as a graph propagation problem with added differential privacy.

Market concentration. Antitrust and competition law require regulating agencies to monitor the concentration of revenue across the participants in many markets. This can be notoriously difficult when private companies – who are under no obligation to publish their revenues – are involved. For example, the Herfindahl-Hirschman index [5], a standard measure of market concentration, requires the (private) market shares of each participant as inputs.[2] MPC allows this computation to be performed without market participants having to disclose their revenue composition.

3. USABILITY GOALS

In earlier work, we integrated MPC with a MapReduce platform [13]. By building atop a workflow manager like Musketeer [4], this work generalizes our approach beyond a specific system (e.g., MapReduce), and achieves several usability benefits.

1. *Code generation is automated:* the participating parties need no in-house MPC implementation or deployment expertise, since Musketeer generates all necessary code automatically.

2. *Portability across data analytics stacks:* different companies can map a high-level joint computation to their individual existing data analytics stacks (e.g., Hadoop, Spark, Naiad) via Musketeer, and have these systems automatically feed data into the MPC.

3. *Automatic framework choice:* since Musketeer's scheduler already supports automated choice of good backends for a computation, we can extend its performance model to pick-and-choose between different MPC paradigms and combine them depending on the operators used.

The key premise of Musketeer is to decouple the specification of data-parallel workflows in a high-level frontend language from their execution in a parallel backend execution engine. Musketeer takes the user's workflow and translates it into a common intermediate representation (IR): a directed acyclic graph (DAG) of operators. From this IR, Musketeer generates code for multiple parallel backend execution engines, and executes the workflow by flexibly choosing and combining them. To achieve the above benefits for computations involving secure MPC, we added prototype support for MPC to Musketeer.

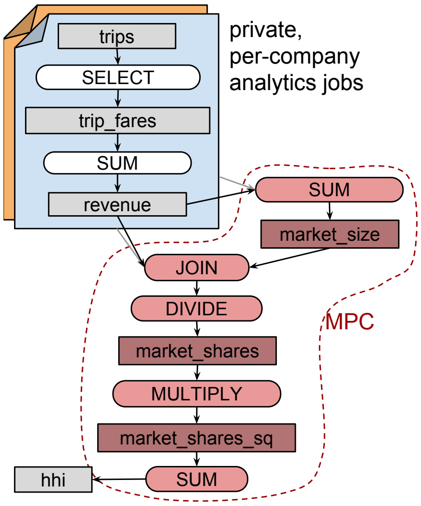

Figure 1: Vehicle-for-hire market concentration workflow: gray boxes are tables, rounded nodes are operators. The red, shaded operations happen in MPC, and arrows crossing the MPC boundary correspond to private inputs.

4. IMPLEMENTATION

We have extended Musketeer with support for input columns to be marked as private, and with a set of MPC operators in the IR. With just these two extensions, Musketeer automatically generates Python code for VIFF's secret sharing-based MPC when given a clique of MPC operators. In the generated code, private columns are secret-shared between parties and computations on them use MPC constructs. Further, we added initial code generation support for the VIFF MPC framework by integrating it as a Musketeer backend. To ensure that secure computations run in MPC, we specified infinite costs for combining them with non-MPC operators and for mapping them to non-MPC backends. We chose VIFF because it is open-source and more general than other available frameworks: it offers a choice between active and passive security as well as adjustable corruption thresholds. In the future, we plan to also include recent and more efficient MPC frameworks as backends. Our Musketeer extensions are open-source, and available at:

`https://github.com/hicsail/Musketeer`.

5. INITIAL RESULTS

We prototyped the market concentration use case discussed in §2 using our implementation. In our example, we compute the Herfindahl-Hirschman Index (HHI) [5] over the market shares of several vehicle-for-hire (VFH) companies. This computation, for example, might allow a regulator to assess the long-term impact of a changing market environment – such as the emergence of "ride-sharing" services such as Uber and Lyft – on market concentration.

The workflow proceeds as shown in Figure 1: each company first computes its local aggregate fare revenue from private trip data using their big data analytics stack of choice. The per-company revenues are then passed into the secure part of the workflow, which sums them under MPC to determine the aggregate revenue, and

[2] The HHI is the sum of squared market shares.

Setup	Runtime
Insecure, trusted Hadoop (8 nodes)	16 min 10 s (970s)
Musketeer with MPC (5 parties, 1+1+1+1+4 nodes)	17 min 31 s (1,051s)
Secure MPC framework only (VIFF only, 5 parties, 5 nodes)	>2 hours (7,200s)

Table 1: End-to-end runtimes for the vehicle-for-hire market HHI computation. Our MPC-extended Musketeer workflow is almost as fast as an insecure analysis on a trusted Hadoop cluster, and much faster than using an MPC framework only.

subsequently computes market shares by dividing each per-company revenue by the total revenue. Finally, the secure MPC computes the HHI by squaring the market shares and adding the results.

We use six years of public NYC taxi trips' fare information [11] as our input data, dividing the data across five imaginary taxi companies (50%/20%/10%/10%/10%). Each company privately computes the initial revenue on between 16 and 80 GB of trip data in their own Hadoop cluster running on Amazon EC2. The results are automatically passed into a shared VIFF cluster with three compute parties, also running on EC2.

Table 1 shows our preliminary results. We compare *(i)* the runtime of this workflow on a single Hadoop cluster operated by a trusted third party (e.g., the regulating authority); *(ii)* the runtime of the same workflow when implemented entirely in Python and VIFF; and *(iii)* the end-to-end runtime for our integrated, mixed-mode MPC Musketeer workflow. Having a trusted third party run this computation is both impractical (must ship hundreds of GB of data) and contentious (the VFH companies might not wish to disclose their per-trip fare information). However, the trusted third party case is a useful performance baseline, since it constitutes the fastest possible execution of this workflow (as using MPC can only add overhead). As our results show, the integrated Musketeer workflow only takes 8.3% longer than this baseline (1,051s vs. 970s), since the data-intensive parts of the computation run in companies' private Hadoop clusters and parallelize well. By contrast, had the companies executed the entire computation in Python and VIFF, it would not have finished after two hours, and required substantial MPC expertise to implement.

6. DEMO

In our demo, we show a Musketeer-based implementation and live execution of the market concentration use case described above. First, we illustrate how an analyst or regulator specifies the joint computation in a SQL-like Musketeer front-end language. Second, we visualize the decomposition into a Musketeer IR DAG, highlighting the parties' input ownership, the boundaries between their local computations, and the joint MPC. Third, we show the generated code executing on real Hadoop and VIFF clusters, and reproduce our evaluation results from §5.

7. FUTURE DIRECTIONS

We are currently extending Musketeer to support other MPC frameworks such as Sharemind [3]. Moreover, Musketeer's scheduler can automatically choose which system is used to execute a particular operator (e.g., based on a simple performance model). We plan to exploit and extend this capacity in several ways to improve the out-of-box performance of MPC.

Static analysis and optimization techniques can similarly be leveraged to help Musketeer pick the most performant MPC implementation for a given workflow, as in work on inferring and improving the performance of MPC protocols [6]. They might also help Musketeer to select an appropriate partitioning, similar to strategies used in work on MPC protocol selection [7].

We will also look at how end-users specify their security and privacy requirements. In real-world scenarios, the authors of an analytics algorithm may not know the privacy requirements of input data contributors. Analysts might require a framework that supports *policy-agnostic programming* [14], in which security and privacy properties are abstracted away from the programmer and specified independently. We are currently working on initial support for this approach by extending Musketeer to automatically detect operations for which data must cross trust domain boundaries, and automatically using MPC for these operations.

Acknowledgements

Research supported by NSF awards #1414119 and #1430145.

8. REFERENCES

[1] VIFF, the Virtual Ideal Functionality Framework. http://viff.dk/. Accessed 01/08/2016.

[2] E. A. Abbe, A. E. Khandani, and A. W. Lo. Privacy-preserving methods for sharing financial risk exposures. *American Economic Review*, 102(3):65–70, May 2012.

[3] D. Bogdanov, S. Laur, and J. Willemson. Sharemind: A Framework for Fast Privacy-Preserving Computations. In *ESORICS*, volume 5283 of *LNCS*, pages 192–206. Springer, 2008.

[4] I. Gog, M. Schwarzkopf, N. Crooks, M. P. Grosvenor, A. Clement, and S. Hand. Musketeer: all for one, one for all in data processing systems. In *EuroSys*, Apr. 2015.

[5] A. O. Hirschman. The paternity of an index. *The American Economic Review*, 54(5):761–762, 1964.

[6] F. Kerschbaum. Automatically optimizing secure computation. In *CCS*, pages 703–714. ACM, 2011.

[7] F. Kerschbaum, T. Schneider, and A. Schröpfer. Automatic protocol selection in secure two-party computations. In *ACNS*, pages 566–584. 2014.

[8] E. Meijer, B. Beckman, and G. Bierman. LINQ: Reconciling Object, Relations and XML in the .NET Framework. In *SIGMOD*, pages 706–706, 2006.

[9] A. Narayan, A. Papadimitriou, and A. Haeberlen. Compute globally, act locally: Protecting federated systems from systemic threats. In *HotDep*, Oct. 2014.

[10] A. Rastogi, M. A. Hammer, and M. Hicks. Wysteria: A programming language for generic, mixed-mode multiparty computations. In *IEEE S&P*, pages 655–670, 2014.

[11] T. W. Schneider. NYC taxi trip data. https://github.com/toddwschneider/nyc-taxi-data. Accessed 03/08/2016.

[12] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[13] N. Volgushev, A. Lapets, and A. Bestavros. Programming Support for an Integrated Multi-Party Computation and MapReduce Infrastructure. In *HotWeb*, Nov. 2015.

[14] J. Yang, T. Hance, T. H. Austin, A. Solar-Lezama, C. Flanagan, and S. Chong. End-to-end policy-agnostic security for database-backed applications. *CoRR*, abs/1507.03513, 2015.

[15] A. C. Yao. Protocols for secure computations. In *FOCS*, pages 160–164, 1982.