Symmetric Queries as a Building Block for Efficient Parallel Query Evaluation

Yuqing Melanie Wu

Pomona College
Claremont, CA

Indiana University
Bloomington, IN

Motivating Examples

Many applications, especially of which data-intensive, have to deal with sequences of sets of objects, where all objects are of the same type…

Same query, different complexity

- B: Boolean
- A: Aggregated value
- I : Object incidence

Example applications:

Q_1. parts that are supplied by at least three suppliers

- Q_1^B: Does there exist a part that is supplied by at least three suppliers?
- Q_1^A: How many parts are supplied by at least three suppliers?
- Q_1^I : Find all parts that are supplied by at least three suppliers.

Q_2. parts that are supplied by all suppliers

Q_3. a set of more than one part that is supplied by the same set of suppliers

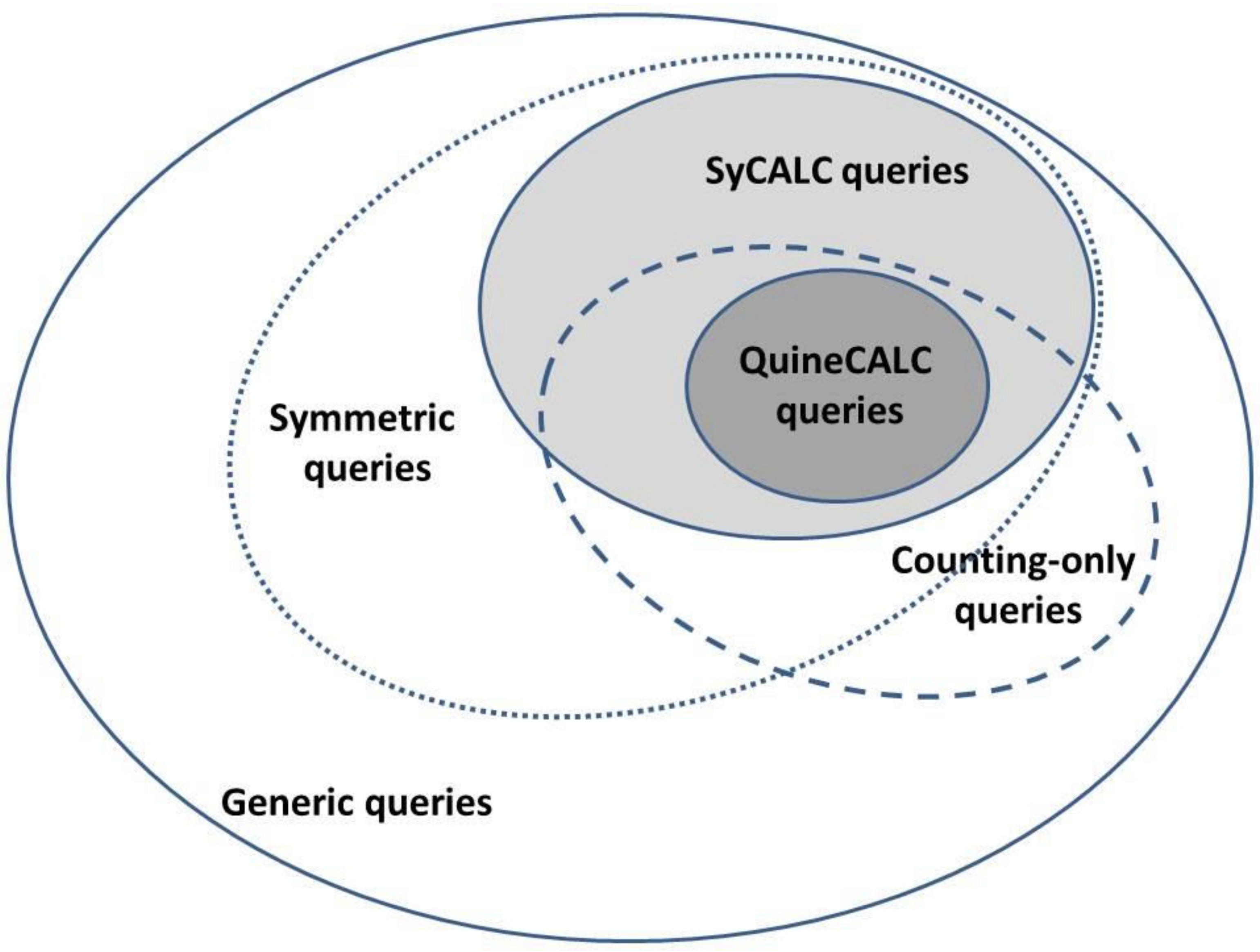
Q_4. pairs of parts such that if a supplier supplies the first one, it must supply the second one

Q_5. pairs of parts such that there is at least one supplier that supplies both

Q_6. pairs of parts such that there is no supplier that supplies both

Q_7. pairs of parts such that there are exactly two-hundred suppliers that supply both

Symmetric Query Languages



Open Questions

1. Is there any summary information or techniques that can help speed up the evaluation of the queries in the Q7 group, sparing it from two hundred self-joins?
2. For the queries whose complexity is between the group of queries in the example, to what degree are their evaluations parallelizable?
3. If some of these queries are not naturally parallelizable, can we find smart evaluation techniques to compute parts of it in a parallel manner?

Broader Questions

1. What are the languages whose queries are naturally parallelizable?
2. How can we identify parallelizable components in a generic query?
3. How can we evaluate a generic query efficiently in a parallel environment?

QuineCALC

$$\varphi \coloneqq \Gamma(x,X)|\ X=Y\ |X\neq Y|\varphi_1 \wedge \varphi_2|\varphi_1 \vee \varphi_2|\neg\varphi_1|\exists X \varphi_1$$

1. A restricted first-order logic
2. A generalization of symmetric n-ary Boolean functions whose arguments and values are sets.
3. All QuineCALC queries are *counting-only.*

SyCALC

$$\varphi \coloneqq \Gamma(x,X)|\ X=Y\ |X\neq Y|\varphi_1 \wedge \varphi_2|\varphi_1 \vee \varphi_2|\neg\varphi_1|\exists x \varphi_1|\exists X \varphi_1$$

1. Extension of QuineCALC, accommodating projection and Cartesian product.
2. For every SyCALC query *q*, for every natural number *n*, there exists a symmetric relational function $f_{q,n}(S_1,\ldots,S_1)$*such that* $q \equiv_n f_{q,n}(S_1,\ldots,S_1)$.

Education

It is not a second thought.

Open questions:

1. Where/how to push the concepts of parallel computing into undergraduate curriculum?
2. What are the general concepts about parallel computing that are the must-have for undergraduate CS major?
3. What is the core/essence of the parallel computing of which different sub-areas have identified their own focuses?
4. How can the concepts of parallel computing be specialized in target areas?

NSF-XPS Workshop, Arlington, VA. June 2015