Повышение производительности языка LaOQL за счет параллельной реализации запросов

В. М. Димитров

Петрозаводский Государственный Университет
Кафедра Информатики и Математического Обеспечения

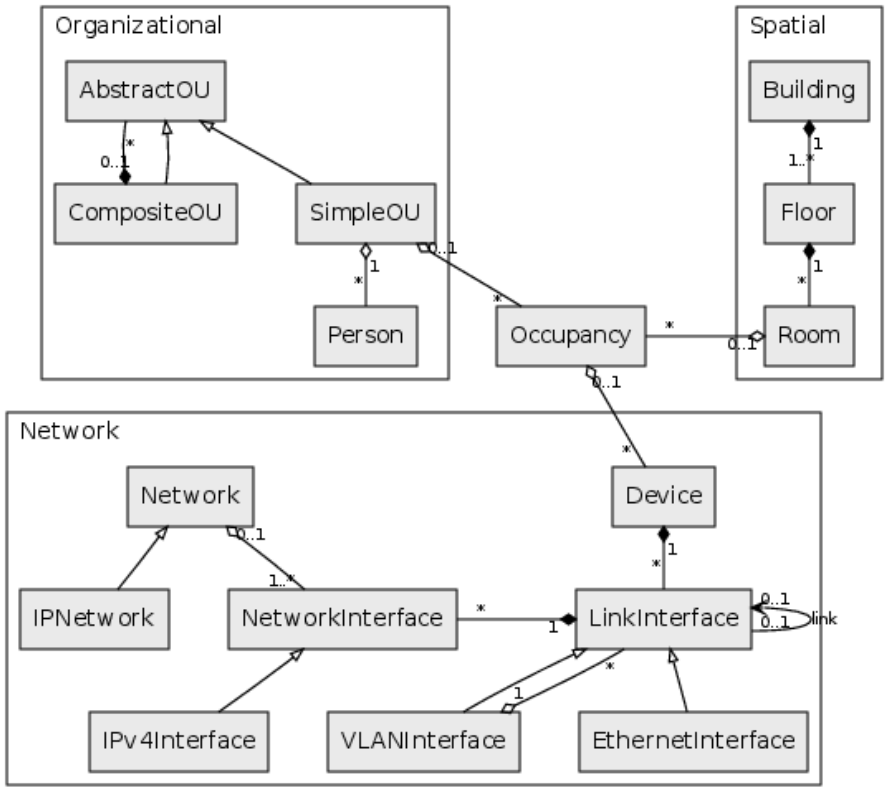
Содержание

- Язык LaOQL.
- Параллельное выполнение запросов.
- Система сравнения производительности.
- Результаты.

Язык LaOQL

Задача

Предоставить пользователю простой инструмент для работы с моделью и источниками дополнительной информации.

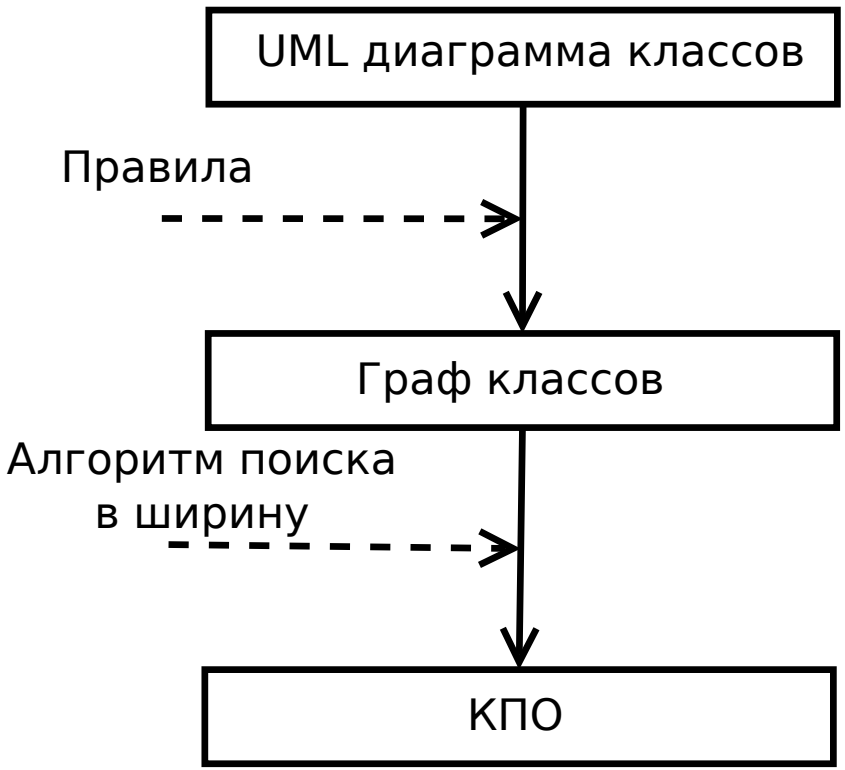


Димитров В. М., Воронин А. В., Богоявленский Ю. А. Лаконичный язык запросов LaOQL на основе связей в объектной модели данных // Программная инженерия. - 2017. - № 3. - С. 112-119.

Основная идея лаконизации текста запроса

Сократить выражения путей:

- Создание структуры путей с помощью алгоритма обхода в ширину на основе графа классов.
- Ручное изменение данной структуры поставщиком системы.



Сокращение текста запроса

- исключение ключевых слов (select, where, from, group by, sort by и др.);
- исключение указания связей между объектами;
- использование сокращенных названий объектов;
- введение понятия свойства по умолчанию;
- сокращенная запись сложных условий;
- вызов функции.

Параллельное выполнение запросов

Параллельность для LaOQL

Возможности для параллельности

- связанные элементы:
  - только от корневых элементов;
  - на любом уровне вложенности.
- вызов функций;
- условия;
- пересечение и объединение;

Технические решения и проблемы

- библиотека Fork/Join;
- надстройка в Clojure — reducers;

Только от корневых элементов $C_1(C_2(C_3))$

Th1:	$O1_{C_1} \rightarrow$	$O11_{C_2} \rightarrow$	$O111_{C_3}$
			$O112_{C_3}$
			$O113_{C_3}$
		$O12_{C_2} \rightarrow$	$O121_{C_3}$
			$O122_{C_3}$
			$O123_{C_3}$
		...	
Th2:	$O2_{C_1} \rightarrow$	$O21_{C_2} \rightarrow$	$O211_{C_3}$
			$O212_{C_3}$
			$O213_{C_3}$
		$O22_{C_2} \rightarrow$	$O221_{C_3}$
			$O222_{C_3}$
			$O223_{C_3}$
		...	
...			

На любом уровне вложенности $C_1(C_2(C_3))$

Th1:	$O1_{C_1} \rightarrow$	Th11:	$O11_{C_2} \rightarrow$	$O111_{C_3}$
				$O112_{C_3}$
				$O113_{C_3}$
		Th12:	$O12_{C_2} \rightarrow$	$O121_{C_3}$
				$O122_{C_3}$
				$O123_{C_3}$
			...	
Th2:	$O2_{C_1} \rightarrow$	Th21:	$O21_{C_2} \rightarrow$	$O211_{C_3}$
				$O212_{C_3}$
				$O213_{C_3}$
		Th22:	$O22_{C_2} \rightarrow$	$O221_{C_3}$
				$O222_{C_3}$
				$O223_{C_3}$
			...	
...				

Инструмент Fork/Join

- Спроектирован для работы на одной Java VM.
- Деление на потоки в рекурсивной манере.
- Возможность коммуникации между потоками

Инструмент reducers (MapReducers)

- Спроектирован для работы на кластере из Java VM.
- Один раз разделить задачу на потоки.
- Нет возможности коммуникации между потоками.
- Новые функции map/reduce/fold.
- Использует обычные структуры данных (не параллельные).
- Реализация без итераций.

Система сравнения производительности

Генерация SON-структуры для системы измерения производительности

- Генерирует структуру с заданным количеством элементов.
- Для определения количества конкретного элемента используются весовые коэффициенты.
- Следующие поля инициализируются случайными значениями из заданного набора:
  - name, description для всех;
  - number для Floor, Room;
  - inetAddress для IPv4Interface;
  - mask, address для IPNetwork;
  - macAddress для EthernetInterface.
- Пригодна для использования в Nest.

Коэффициенты для SON-элементов

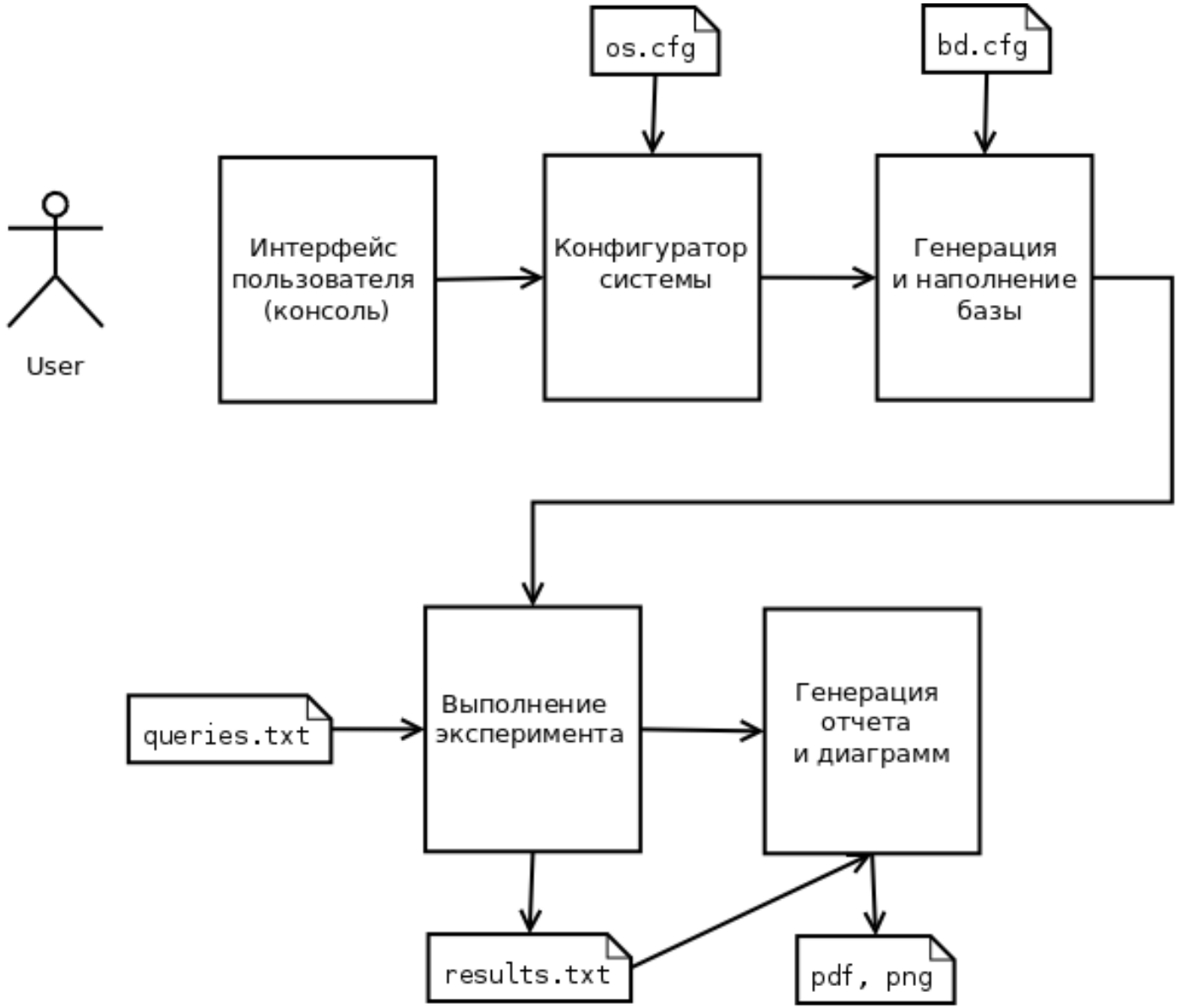
Building	1
Floor	5
Room	50
Occupancy	60
SimpleOU	20
CompositeOU	5
Device	300
NetworkInterface	300
EthernetInterface	300
IPv4Interface	300
IPNetwork	5
VLANInterface	20

* Подобрано на основе данных о реальных количествах сети ПетрГУ

Сценарии и запросы для оценки

- Реально используемые сценарии запросов:
  - Скрипт sonEnlivener.js: наполнение SON-структуры после сбора (132 запроса для собранной структуры ПетрГУ).
  - Плагин AddressInfo: получение информации по заданному IP-адресу (6 запросов).
  - SonNestTree: список запросов, использующихся для построения дерева (4 запросов).
- Список отдельных запросов:
  - Запрос на соединение: device (building).
  - Запрос на соединение: building (device).
  - Запрос на соединение: linkinterface (network (device)).

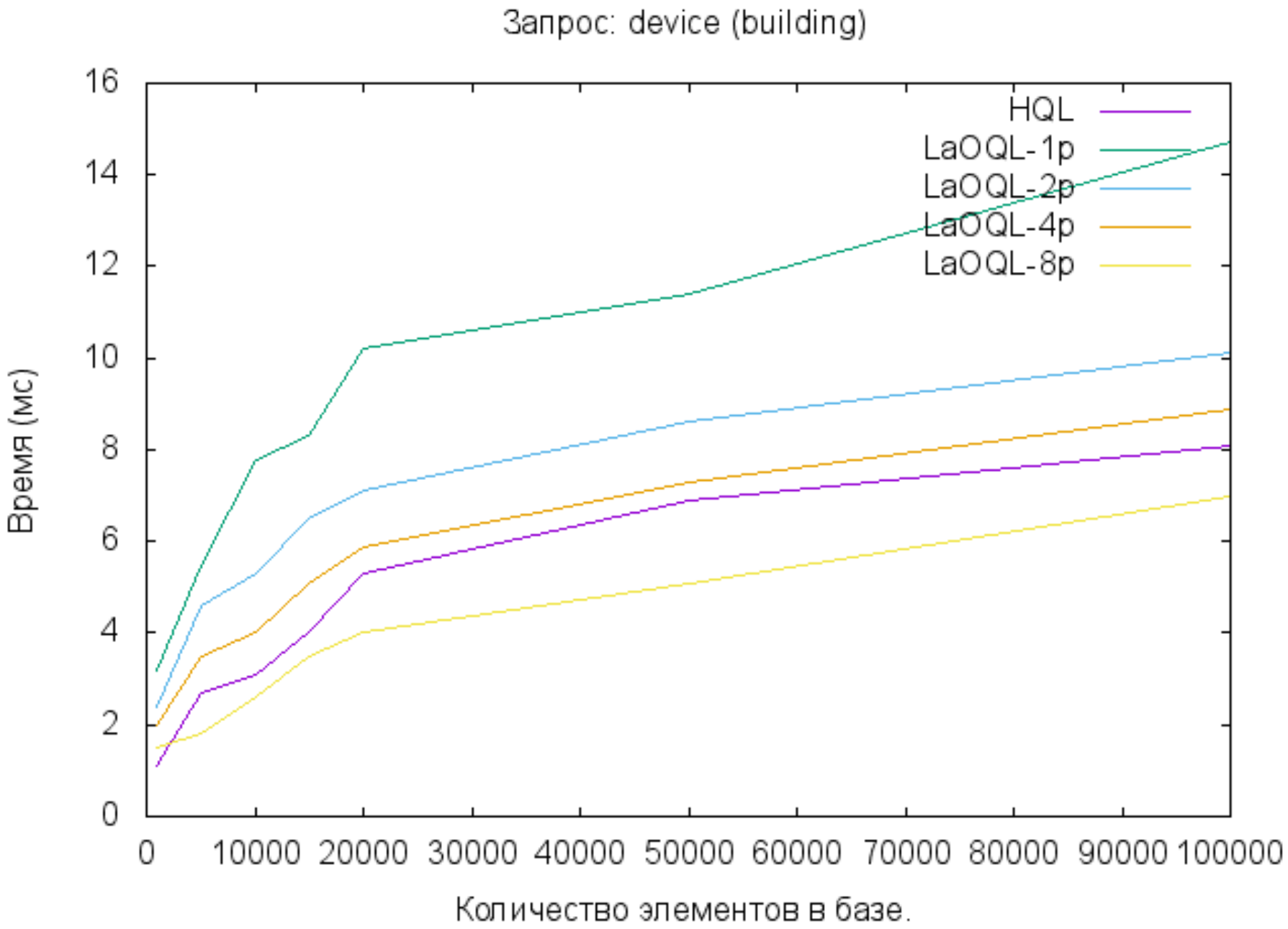
Архитектура

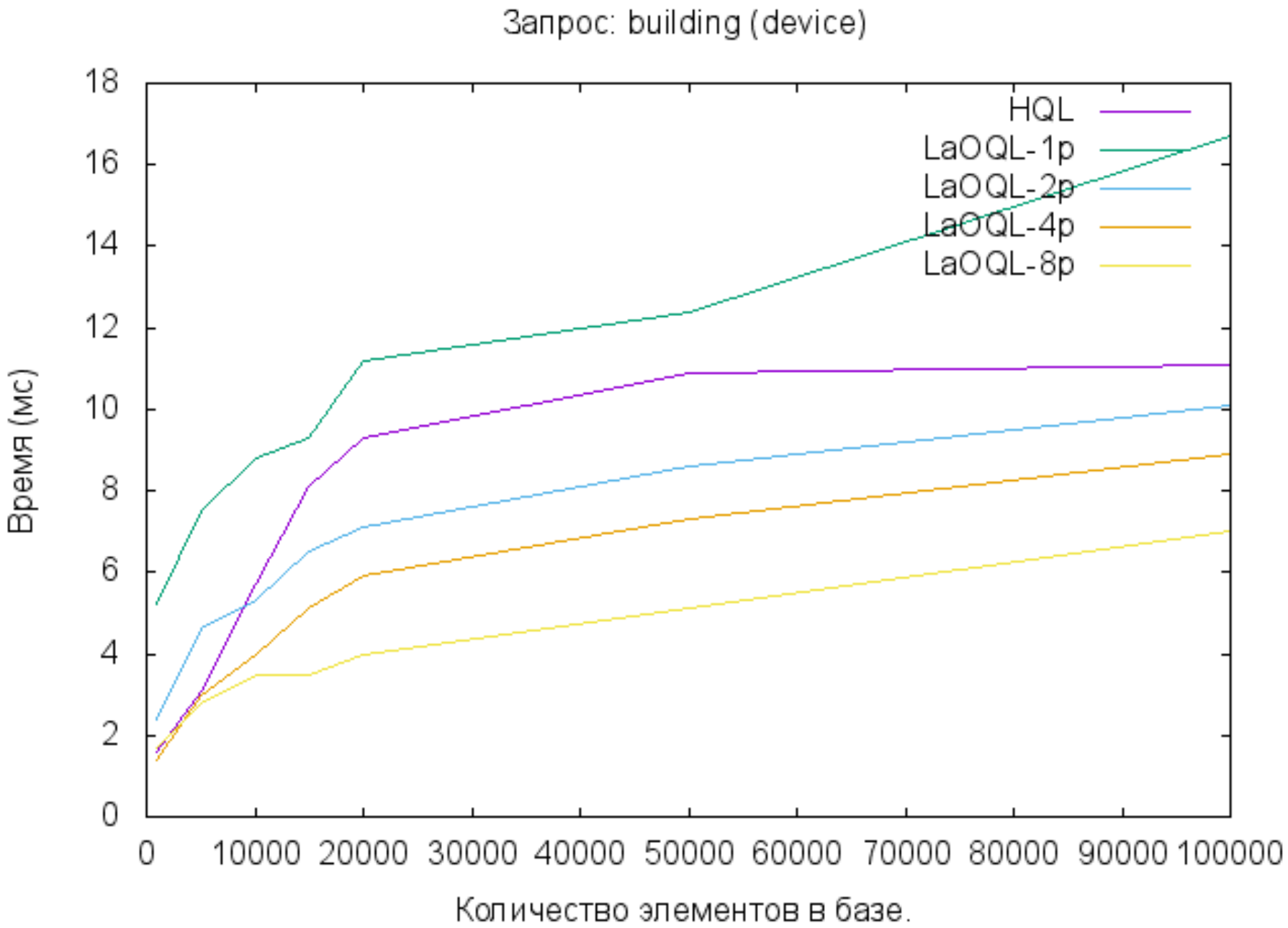


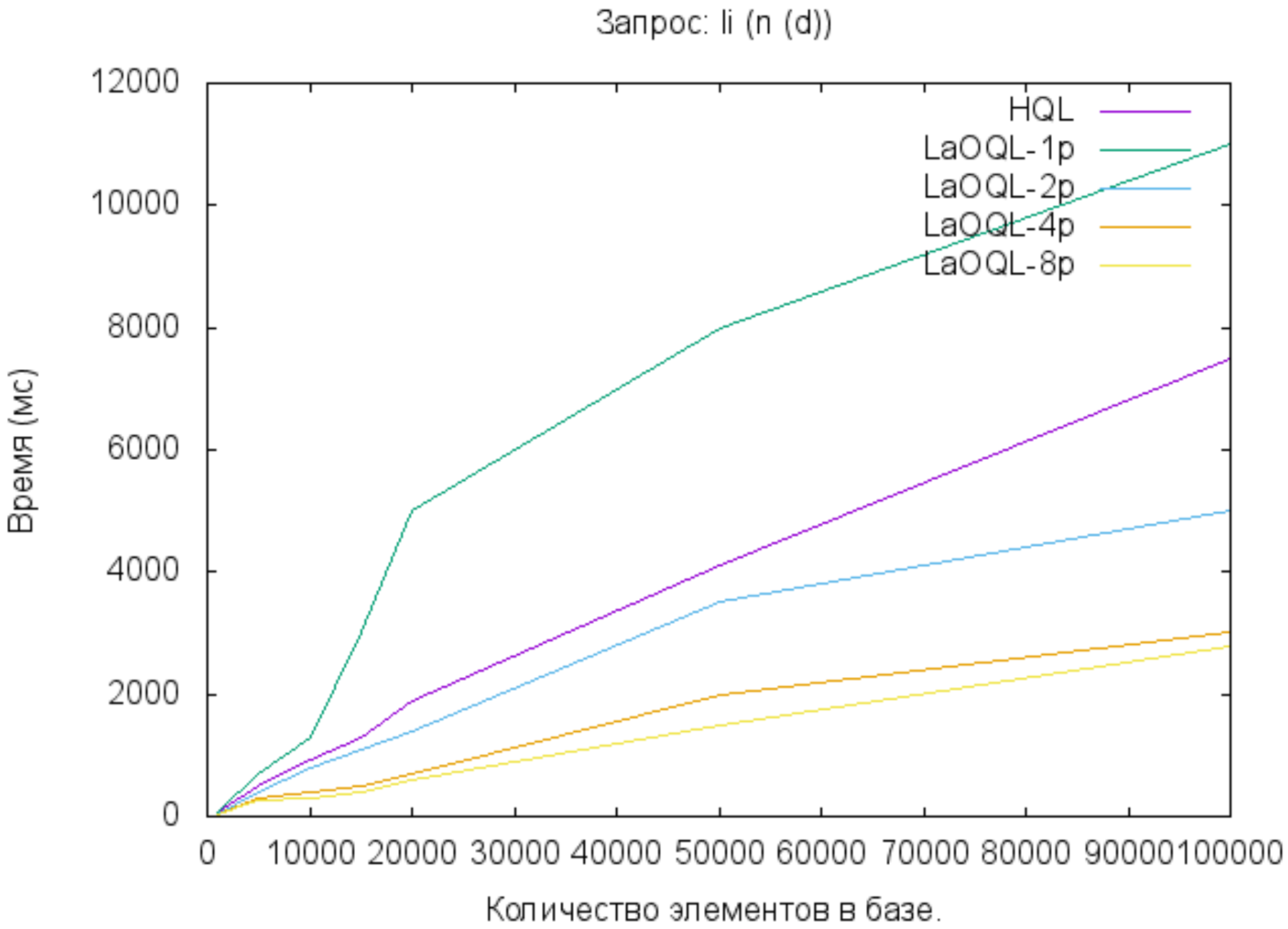
Эксперименты

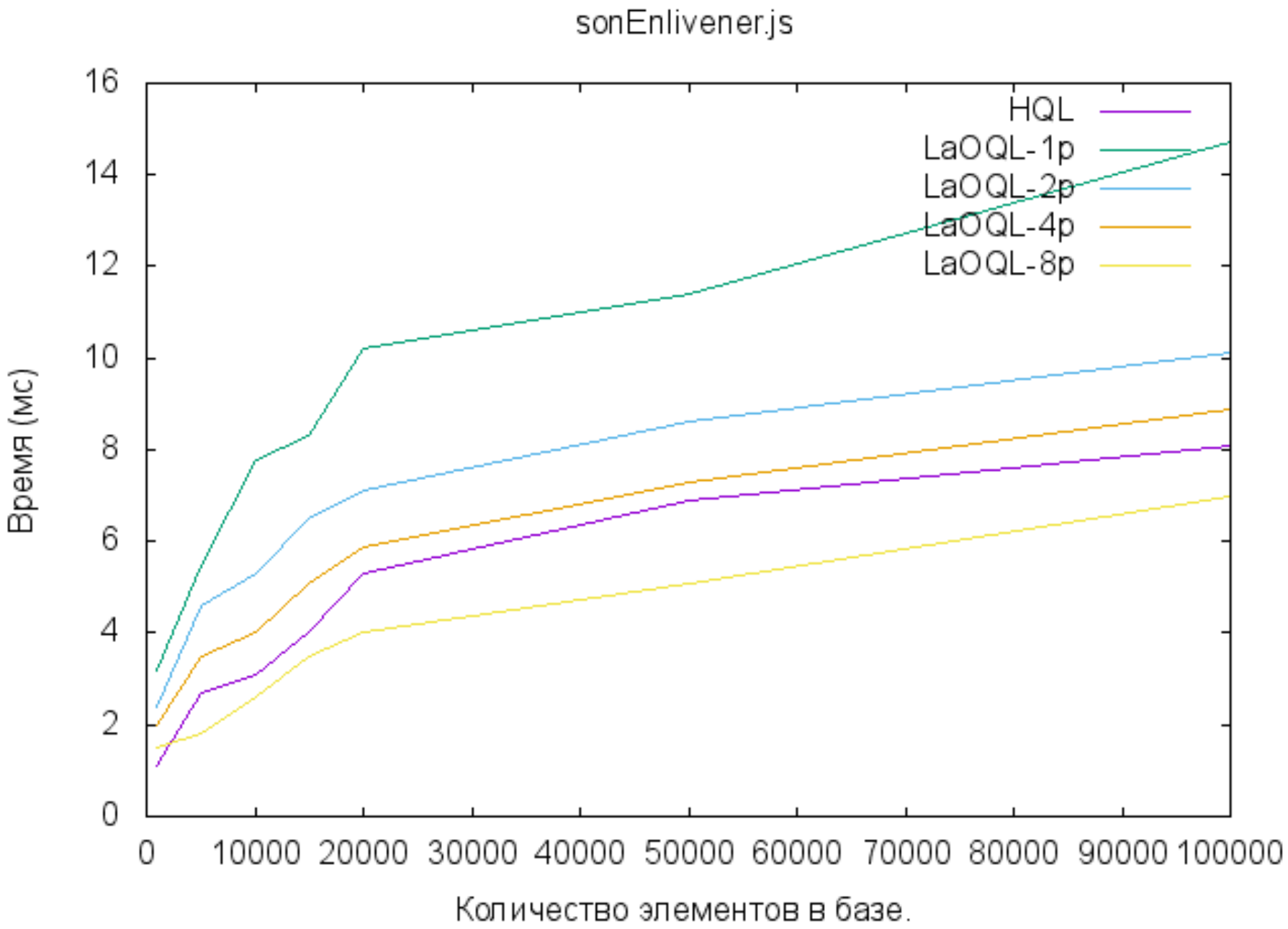
- Конфигурация: Intel Xeon, ОЗУ 4 Гб, ОС Linux.
- Языки: LaOQL, HQL.
- Базы данных: H2, Derby, HSQLDB, Store.
- Тип памяти: оперативная, долговременная.
- Количество элементов в БД: 1000, 5000, 10000, 15000, 20000, 50000, 100000.
- Количество запусков: 10
- Измеряемые характеристики: время работы запроса, время cpu потока, время user потока, потребление памяти.
- Вычисляемые статистические величины по полученным выборкам: среднее, 5%, 50% и 90% квантилю.
- Построение диаграмм происходит по 50% квантилю.
- Запуск теста с “разминкой”.

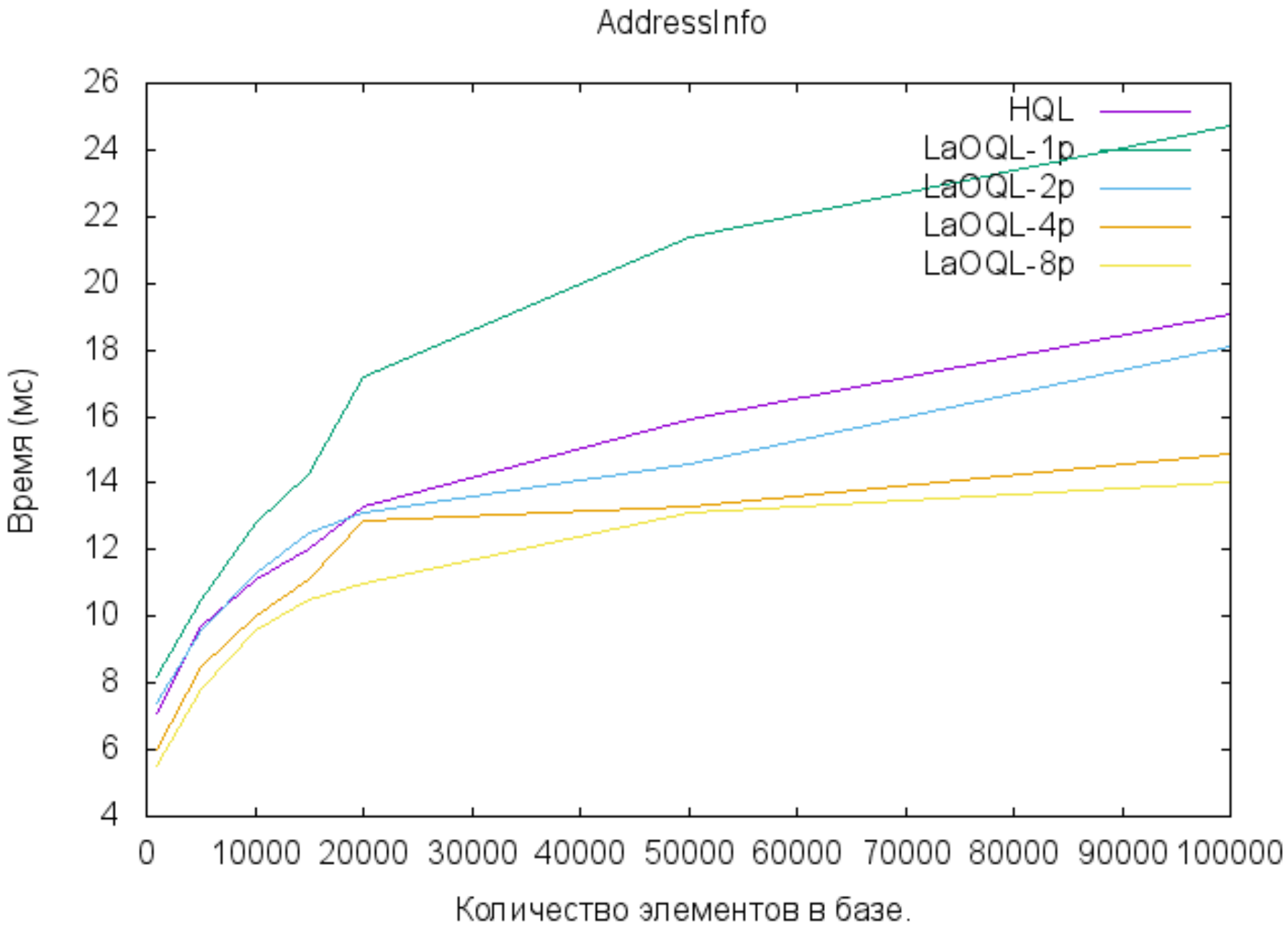
Результаты

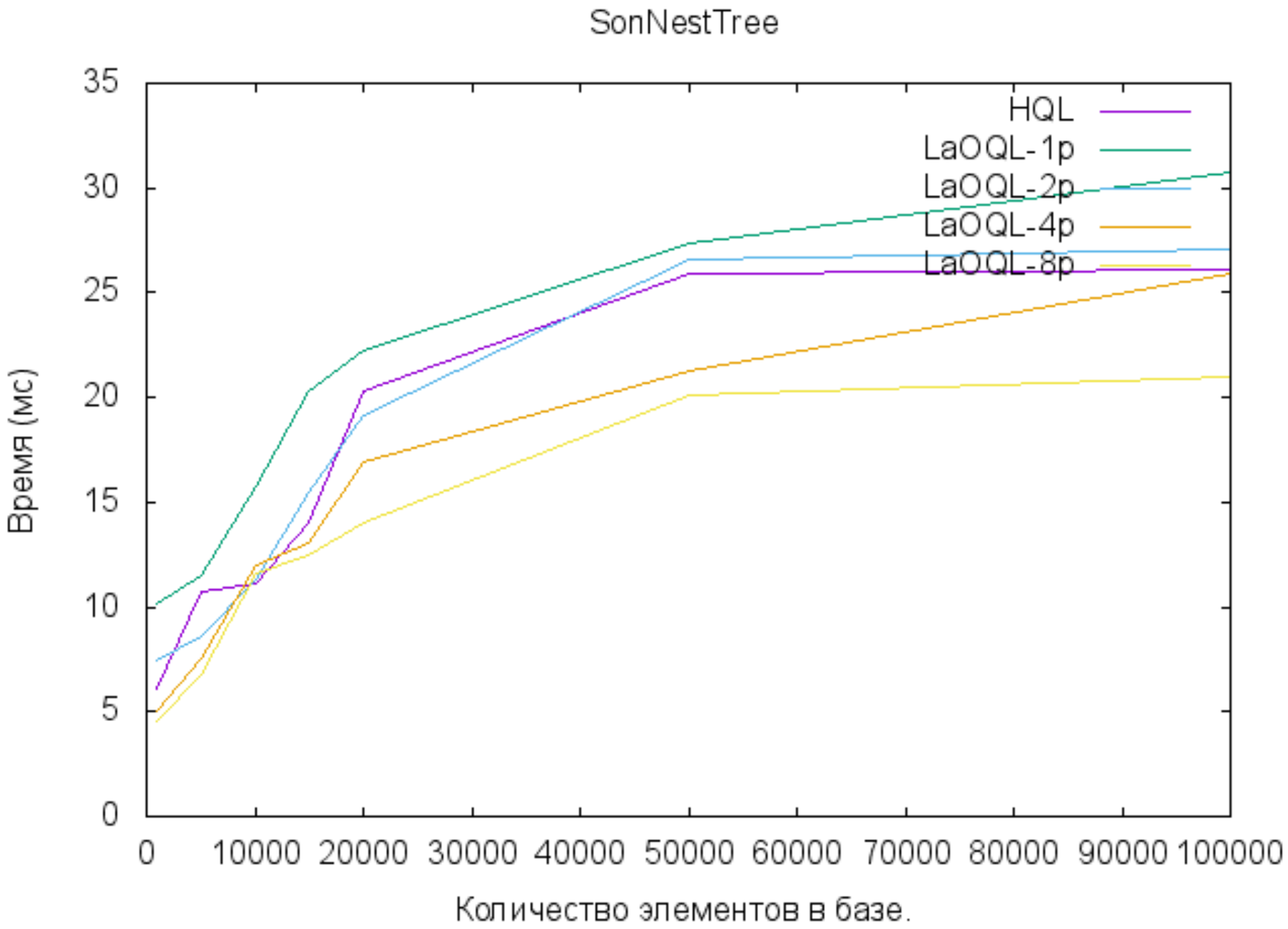

Запрос: device (building)
Время (мс)
Количество элементов в базе.
HQL
LaOQL-1p
LaOQL-2p
LaOQL-4p
LaOQL-8p
0
2
4
6
8
10
12
14
16
0
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000


Запрос: building (device)
Время (мс)
Количество элементов в базе.
HQL
LaOQL-1p
LaOQL-2p
LaOQL-4p
LaOQL-8p
0
2
4
6
8
10
12
14
16
18
0
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000


Запрос: li (n (d))
Время (мс)
12000
10000
8000
6000
4000
2000
0
0
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000
Количество элементов в базе.
HQL
LaOQL-1p
LaOQL-2p
LaOQL-4p
LaOQL-8p


sonEnlivener.js
HQL
LaOQL-1p
LaOQL-2p
LaOQL-4p
LaOQL-8p
Время (мс)
0
2
4
6
8
10
12
14
16
0
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000
Количество элементов в базе.


AddressInfo
HQL
LaOQL-1p
LaOQL-2p
LaOQL-4p
LaOQL-8p
Время (мс)
26
24
22
20
18
16
14
12
10
8
6
4
0
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000
Количество элементов в базе.


SonNestTree
HQL
LaOQL-1p
LaOQL-2p
LaOQL-4p
LaOQL-8p
Время (мс)
0
5
10
15
20
25
30
35
0
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000
Количество элементов в базе.

Результаты

- Разработаны и реализованы сценарии для сравнения производительности и потребления оперативной памяти языком LaOQL с существующими средствами извлечения данных из хранилища.
- Проведен сравнительный анализ производительности и потребления оперативной памяти языком LaOQL с существующими средствами извлечения данных из хранилища.
- Реализованы и внедрены методы распараллеливания выполнения запроса в язык LaOQL.
- Проведен сравнительный анализ результатов полученных до и после внедрения параллельности.

Спасибо за внимание!